COMPAQ

PORTABLE COMPUTER

DISK OPERATING SYSTEM

REFERENCE GUIDE

NOTICE

The information contained in this manual is subject to change without notice.

COMPAQ Computer Corporation shall not be liable for technical or editorial omissions made herein; nor for incidental or consequential damages resulting from the furnishing, performance, or use of this material.

DISK OPERATING SYSTEM REFERENCE GUIDE

First Edition (December 1982)
Second Edition (May 1983)
Third Edition (August 1983)
Assembly #100004-001
Text Subassembly #0340-100321-001

COMPAQ Computer Corporation

COMPAQ
COMPAQ
COMPAQ
COMPAQ
COMPAQ
COMPAQ
COMPAQ™

PORTABLE COMPUTER

DISK OPERATING SYSTEM

REFERENCE GUIDE

SCOPE AND PURPOSE OF THE DISK OPERATING SYSTEM REFERENCE GUIDE

This manual presents the capabilities of the COMPAQ Computer disk operating system (DOS). This system permits you to create, edit, link, debug, and execute programs.

Reading and firmly understanding the material in the COMPAQ Portable Computer OPERATIONS GUIDE is crucial to your understanding of this manual. Much of the critical information presented in that manual is not covered in this text.

ORGANIZATION OF THIS MANUAL

Chapter 1, INTRODUCTION TO THE DISK OPERATING SYSTEM (DOS): This chapter describes DOS in terms of its functional use.

Chapter 2, SYSTEM RESOURCES: This chapter provides background information about system requirements for implementation of DOS.

Chapter 3, DOS COMMANDS: Presents detailed descriptions of the DOS commands. The discussion covers the following topics: general command parameters, batch processing, and specific DOS commands.

Chapter 4, EDLIN: This chapter gives a detailed explanation of commands used to create, alter, and display source language files and standard text files.

Chapter 5, DEBUG: The objective of this chapter is to arm you with the commands and understanding of the DOS DEBUG program. This section discusses how to monitor and control the execution of a program that needs debugging.

Chapter 6, LINK: The guidelines found in this chapter assist you in linking various programs together to enlarge the potential of your applications.

Appendices: The various appendices are intended for reference in the use of DOS. Information includes messages generated by the terminal, general technical information, diskette space allocation, system interrupts, and more.

ASSOCIATED DOCUMENTATION

From time to time, you may wish to refer to one of the following system documents:

OPERATIONS GUIDE (#100000-001): A comprehensive guide to user orientation. This manual provides an introduction to DOS operations, BASIC programming, user diagnostics, and various options.

BASIC REFERENCE GUIDE (#100005-001): A guide to programming in the BASIC language. This manual provides user orientation to special editing keys, the filing system, constants, variables, and miscellaneous expressions.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1-1. INTRODUCTION

This manual gives an overview of the COMPAQ Computer disk operating system (DOS), provided on the DOS diskette packaged in the OPERATIONS GUIDE. It describes the operator interface, the file system, the command structure, and the available commands. It also contains chapters on:

EDLIN.COM, the DOS line editor
DEBUG.COM, the DOS debug utility
LINK.EXE, the LINK linker

1-2. FEATURES AND BENEFITS OF COMPAQ COMPUTER DOS

Device Independent Input/Output (I/O)

COMPAQ Computer DOS simplifies I/O to different peripheral devices by assigning a reserved filename to each device. These names are built-in to DOS and are detected by the DOS file system. Programs designed only for disk file I/O can receive input from the keyboard or send output to the video display or line printer.

Advanced Error-Recovery Procedures

COMPAQ Computer DOS does not necessarily require rebooting when diskette errors occur. If a diskette error occurs at any time during any program, DOS retries the operation three times. If the operation cannot be completed successfully, DOS returns an error message, then waits for you to enter a response. You can attempt to recover from the error before rebooting the operating system.

Flexible File Characteristics

In COMPAQ Computer DOS, there is no practical limit on file or disk size.

DOS remembers the exact physical end-of-file marker. Should a file be opened with a logical record length greater than the physical record length, DOS remembers exactly where the file ends to the byte, rather than rounded to 128 bytes.

Fast, Efficient File Structure

COMPAQ Computer DOS employs a highly-efficient diskette structure which eliminates the need for manual file size extension

operations, minimizes access to the directory track, and provides for duplicate directory information and verifications after writes.

No Need to Log in Diskettes

As long as no file is currently being written, diskettes can be exchanged without logging in. One benefit of this feature is that the DOS debug utility, DEBUG.COM, can be used without reloading for different programs on different diskettes.

Time and Date Stamps

If the current date and time are entered during system start-up, DOS automatically records the correct time and date of file activity during the session.

1-3. PROVIDED SOFTWARE

Some of the software provided with the COMPAQ Computer disk operating system is described below. Detailed information of other DOS commands is given in Chapter 3.

CHKDSK.COM is a command used to check and verify the contents of a diskette. It is further described in Chapter 3, DOS COMMANDS.

COMMAND.COM is the command interpreter used to interface between the operator and the underlying operating system. It allows you to perform file management functions, such as renaming and deleting, as well as loading and excuting programs.

DEBUG.COM is a debug utility that provides a controlled testing environment for executable object files. DEBUG.COM is further described in Chapter 5, DEBUG.

EDLIN.COM is the DOS line editor. Intraline editing is performed using the special editing keys. EDLIN.COM is further described in Chapter 4, EDLIN.

EXE2BIN.COM is used to convert .EXE files to .COM files. In general, only assembly language programs that have been specially formulated may undergo such conversions. EXE2BIN.COM is further described in Chapter 3, DOS COMMANDS.

FORMAT.COM is used to format diskettes so that they can be used with DOS. FORMAT.COM is described in Chapter 3, DOS COMMANDS.

LINK.EXE is the LINK linker used to link object files and object libraries to create executable .EXE files. LINK.EXE is further described in Chapter 6, LINK.

IOSYS.COM is the lowest level of the DOS operating system, interfacing to all I/O devices. It is a DOS hidden file and does not show up when a directory command is executed. IO.SYS is automatically loaded into memory when the system is booted up.

MSDOS.COM is the heart of the operating system where most management of system resources takes place. DOS.SYS is intimately tied to COMMAND.COM and IO.SYS. DOS.SYS is a hidden file and does not show up when a directory command is executed. DOS.SYS is automatically loaded into memory when the system is booted up.

SYS.COM is used to transfer DOS.SYS and IO.SYS from a system diskette to a formatted diskette that does not contain the operating system. It is further described in Chapter 3, DOS COMMANDS.

1-4. APPLICATIONS PACKAGES

When you purchase applications packages, such as VisiCalc™, the manufacturer supplies you with complete instructions for use of the program. Should those instructions conflict with the instructions in this manual, use theirs.

2-1. DISKETTES AND DISKETTE SPACE

Space on a particular diskette is divided into files, as shown in Figure 2-1.

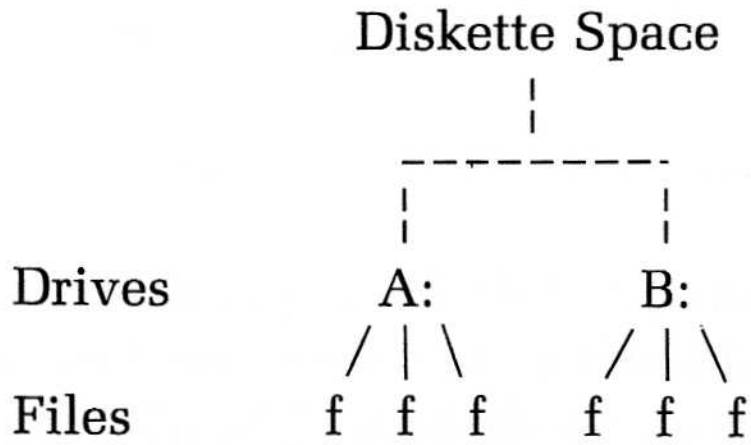


Figure 2-1. Allocation of File Space on a Diskette.

Each double-sided, double-density diskette used with the COMPAQ Computer contains 320 Kbytes of storage space. The DOS diskette, or any copy of it, has its space divided into five parts:

- The Boot Program (Side 0, Track 0, Sector 1)

  This sector contains the boot program which the hardware loads on power up or when ^ALT+DEL is pressed. It will load the rest of DOS.

- Two copies of the File Allocation Table (Side 0, Track 0, Sectors 2-3)

  These tables are used to allocate disk space to files. DOS maintains these areas.

- Directory (Side 0, Track 0, Sectors 4-8 and Side 1, Track 0, Sectors 1-2)

  The directory contains information about each file on the diskette, including the file specification, size of the file, and the time and date of last modification.

- IOSYS.COM (Side 1, Track 0, Sectors 3-6)

  This is the interface program used by DOS to communicate I/O information to the COMPAQ Computer hardware system.

- MSDOS.COM (Side 1, Track 0, Sectors 7-8; Side 0, Track 1, Sectors 1-8); and side 1, Track 1, Sectors 1-4)

  This is where the operating system resides.

- Data Area (Starts Side 1, Track 1, Sector 5)

  The greater portion of diskette space is reserved for the contents of files. An individual file does not necessarily reside in continuous sectors on the diskette. The file allocation table is used to allocate diskette sectors to files.

  Usually the COMMAND.COM program is the first file in the data area but this is not required.

2-2. DOS DISKETTE BACK-UP

The DOS diskette is essential to operation. Most importantly, it has a protective adhesive tab over the write-protect notch and should not be written to or used for storage.

For these reasons, it is very important that the diskette be backed up with one or more copies, and the original stored in a safe place where it will not be exposed to static electricity, magnetic fields, extremes of temperature or humidity, or subjected to excessive handling or other abuse.

Copies of the DOS diskette can be made using the DISKCOPY command. DISKCOPY copies the entire diskette onto another diskette.

With a Dual-Drive System

To copy DOS from the original (source) diskette to the new (target) diskette using DISKCOPY, enter:

```
A> DISKCOPY A: B:
```

The screen displays:

```
Insert source diskette in drive A:

Insert target diskette in drive B:

Strike any key when ready
```

If the diskettes are not already in place, insert them. Once they are inserted, press any key to begin. Blocks of data will be transferred from the source diskette to memory and then moved to the target

diskette. This process is repeated until the copy is complete. When finished, the screen displays:

```
Copy complete

Copy another? (Y/N)
```

Press N to terminate the copy session.

With a Single-Drive System

The DISKCOPY command is the same in principle whether your system has one disk drive, or two. The chief difference is one of time and effort. When you enter:

```
A> DISKCOPY A: B:
```

the screen displays:

```
Insert source diskette in drive A:
Strike any key when ready.
```

Insert the source diskette if you have not already done so. Press any key. The computer reads as much information as it can accommodate into memory, then requests:

```
Insert the target diskette in drive A:
Strike any key when ready
```

Replace the DOS diskette in the drive with the new diskette. Press any key. The data taken from the DOS diskette is transferred from memory to the new diskette, then the system asks for the source diskette again. This process continues, copying from and onto alternating diskettes, until the copy is complete. The screen displays:

```
Copy complete

Copy another ? (Y/N)
```

The same procedures apply to all commands and activities requiring two diskettes, such as DISKCOMP (to compare diskettes). It is a good idea to compare the DOS diskette and the new copy at this time. Enter:

```
A> DISKCOMP A: B:
```

If you have one disk drive, you will be directed to exchange diskettes in the drive as in the DISKCOPY procedure. If you have two disk drives, the switching is automatic. When the comparison is complete, the screen displays:

```
Diskettes compare ok
Compare another? (Y/N)
```

The Default Drive

At system start-up, the prompt is always A>, indicating the default drive. To specify another, simply type the desired specifier, followed by a colon. Press ENTER.

```
A>B:
```

Note the new prompt identifies the default drive.

```
B>
```

Reading from, or writing to, the diskette in Drive B: does not change the default drive. For example:

```
A>DIR B:
```

displays the directory on the diskette in Drive B:, but the system returns to A> for its next command.

```
A>DISKCOPY A: B:
```

copies the contents of the diskette in Drive A: (or diskette A:) onto the diskette in Drive B: (or diskette B:), but it does not change the default drive.

You can also copy the contents of the diskette in Drive B: to A: by typing:

```
A>DISKCOPY B: A:
```

This does not change the default drive, either. To repeat, only:

```
A>B:
  or
B>A:
```

changes the default drive.

3-1. INTRODUCTION

COMMAND.COM serves as the system command interface between the operating system and operator input. Command interpretation begins when COMMAND.COM scans the operator input for a legal command. COMMAND.COM identifies, interprets, and executes entered commands.

DOS commands are either internal or external. Internal commands are resident in memory as part of COMMAND.COM, and are loaded into memory when the operating system is booted up.

External commands reside on diskettes as program files. Since these commands must be read before execution, a slight delay will occur.

Filenames with extensions of .COM and .EXE are considered to be external commands. They permit you to develop custom applications.

External commands are executed by entering the name of the file without the .COM or .EXE extension. User-created programs in most languages are .EXE files. .EXE files can usually be converted to .COM files with the command, EXE2BIN.

Rules of Syntax

The following syntax rules indicate how the DOS commands must be formatted:

Command Name An extension of .COM or .EXE indicates an external command to DOS. When you enter an external command, do not include the filename extension.

Parameters Parameters included in DOS commands specify additional information to the system. While some parameters are required, some remain

optional. Use the following parameters in user DOS command statements:

Parameter:	**Definition:**
d:	Specifies a disk drive. Identify the drive letter and follow it with a colon (A: or B:).
filename	Specifies the name of the file. The file name includes the filename (which can be a maximum of eight characters). and an optional extension (with as many as three letters). See the .*ext* parameter.

The following are valid characters in file names:

A-Z	$	&	#	@	!
0-9	%	'	(	)	-
	<	>	{	}	_
	\	^	~	¦	`

Any other characters are invalid, and will cause the file name to be truncated (cut short). See the Reserved Device Names section in this chapter.

.*ext*	You can further catagorize your files into smaller, more manageable pieces. You can do this through the use of extensions. These extensions consist of a period (.) followed by a maximum of three characters. Place the extensions immediately after the filename (for example, DOSMAN.1,

which can stand for the DOS manual, Section 1).

The same characters that are valid for filenames are valid for extensions (see filename). Any other characters are invalid.

When referring to a filename with an extension, always include the extension in the file name; otherwise, DOS is unable to locate the file.

Reserved Device Names

DOS reserves the names listed in Table 3-1 for specific system input/output (I/O) devices:

Table 3-1.
Reserved Device Names

Reserved Name:	Device:
CON:	Refers to keyboard input and screen output.To end CON: when using it as an input device, press F6 and then ENTER to create the end-of-file (EOF).
AUX: or COM1	Refers to the first comunications adapter port.
LPT1: or PRN:	Refers to the line printer (output device only).
NUL:	Refers to a dummy (nonexistent) device used for testing various applications. Use NUL: when you do not want to

create a particular file, but the syntax of a command requires an input or output filename.

NOTE:

- Reserved device names may be used in place of file names.
- Drive specifiers or extensions are ignored when used with these device names.
- A colon following the reserved device name is optional.

Wild Card Characters

Use of the wild card characters, * and ? , provides a short-hand notation for multiple file applications. These characters may be used with filenames and their extensions to provide enhanced DOS command flexibility.

A ? character in a filename or extension indicates that any valid character may occupy that position. For example:

```
DIR BUD?DEPT.'83
```

lists all directory entries (in the default drive) beginning with BUD, having any next character, and followed by DEPT, with an extension of '83.

The following files might be listed by this DIR command:

```
BUD1DEPT.'83
BUD2DEPT.'83
BUD3DEPT.'83
```

This listing could indicate that departments 1, 2, and 3 have their 1983 budgets listed on your diskette.

The use of the * character in a filename or extension indicates that any valid character may occupy that position and all remaining positions, including extensions. For example:

DIR D115*.'83

lists all directory entries (in the default drive) beginning with D115, having any next characters, with an extension of '83.

The following files might be listed by this DIR command:

D115LAB.'83
D115EXP.'83
D115FCST.'83

This listing could indicate that department 115 has labor (LAB), expense (EXP) and monthly forecast (FCST) reports for 1983 ('83) listed on this diskette.

The following are example uses of the ? and * characters:

Example 1:

To delete the directory entries for all files named FORECAST on Drive A: (regardless of extension), use the DEL command like this:

DEL A: FORECAST.???

or

DEL A: FORECAST.*

The entries deleted might look like this:

FORECAST.115
FORECAST.117
FORECAST.250
FORECAST.300

The files listed might indicate which departments have submitted their monthly forecasts.

Example 2:
To list the directory entries for all files on Drive B: (regardless of file names) with an extension of JUN, enter:

```
DIR B: ????????.JUN
```

or

```
DIR B: *.JUN
```

The entries listed might look like this:

```
ACCTSREC.JUN
ACCTSPAY.JUN
FORECAST.JUN
EXPENSES.JUN
```

This listing might indicate what monetary figures have been entered for the month of June.

Example 3:
To list the directory entries for all files on Drive B: beginning with 100, and with extensions beginning with 1Q, enter:

```
DIR B: 100?????.1Q?
```

or

```
DIR B: 100*.1Q*
```

The entries listed might indicate for example, that department 100 has the following files:

```
100EXP.1Q     (expense totals for 1Q)
100MAN.1Q     (manpower requirement totals
               for 1Q)
```

100MAN.1QJ (manpower requirements for Jan)
100MAN.1QF (manpower requirements for Feb)
100MAN.1QM (manpower requirements for Mar)

3-2. DOS COMMAND RELATIONSHIPS

File Name Syntax

- The three parts of a complete file name (*d:filename.ext*) must not be separated by characters other than the colon (:) and period (.).

- Filenames are not required to have extensions. If a filename has an extension, that extension must always be included when referring to the file.

- Wild card filenames and device names are valid as command parameters only.

Execution and Control of Commands

- Commands are effective only after pressing the ENTER key.

- To abort a command during execution, press ^BREAK. The command cancels as soon as the system tries to read from the keyboard or display a character on the screen.

- To suspend output of a command, press ^NUM LOCK. This is particularly valuable when the command will produce a large output.

NOTE: In this book, there is a special symbol used to indicate control characters. For example, if the letter T is to be used as a control character, it is written as ^T. The caret (^) superscript always means that you must first press and hold down the CTRL key, and then press the designated character key. ^T is read aloud as "Control T".

Drive Control

The prompts A> and B> indicate which drive is being used as the default drive.

Syntax Notation

All DOS commands in this manual will be formatted for ease of use as follows:

- Commands and file names may be typed in either upper- or lower-case.

- Items in brackets, [ ], are optional.

- Punctuation such as commas, equals signs, question marks, colons, or slashes, must be included where shown.

- You **must** separate commands and parameters with delimiters. A delimiter is a space, comma, semicolon, equals sign, or tab. For example:

  DEL Goodfile.rel;badfile.rel

  or

  COPY,fourfile fivefile

  A space as a delimiter makes the command easier to read. In this book, the usual delimiter is a space.

The COMPAQ Portable Computer permits you to make errors, then recover from those errors without any loss of text. If a typing error is made, backspace, enter the correction, and continue to input.

3-3. BATCH PROCESSING

Type: Internal.

Purpose: To execute the commands found in the specified file from the default or designated drive.

Syntax: [*d*:] *file name* [parameters]

Comments: Batch files contain one or more commands that DOS executes one at a time. All batch files must have a .BAT extension.

When executing a batch file, parameters may be passed to the *filename* .BAT file. Passing parameters permits a single batch file to do similar work with different data during each execution.

Batch files are created through the use of the DOS line editor (EDLIN), or by using the COPY command from the keyboard.

The last command in a batch file may be the name of another batch file. This technique permits you to continue processing.

The AUTOEXEC.BAT File

When powered up or restarted, the computer searches for the AUTOEXEC.BAT file on either the DOS or default diskette. Once found, the system automatically executes commands found in this file.

For example, you can automatically load BASIC and execute the program called PAYABLES at power-up by modifying the AUTOEXEC.BAT file, as follows:

Type:

COPY CON: AUTOEXEC.BAT	This statement tells DOS to copy input from the keyboard to the AUTOEXEC.BAT file.
Type:	
BASIC PAYABLES	This statement is inserted in the AUTOEXEC file.
Press F6, and then press ENTER.	The system displays: 1 File(s) copied indicating the termination of the COPY command from CON; by pressing F6 and ENTER.

NOTE: You can enter either single commands or a series of commands in the AUTOEXEC.BAT file.

When you modify the AUTOEXEC.BAT file, DOS does not prompt you for the date and time unless you include the DATE and TIME commands in the AUTOEXEC.BAT file.

Executing a .BAT File With Dummy Parameters

Consider the following batch file:

```
REM START TEST BATCH FILE
COPY %1 %2
TYPE %2.PRN
TYPE %0.BAT
```

To execute the ASMFILE.BAT file and pass the parameters, enter the batch file name, followed by the parameters to be sequentially substituted for %1, %2, and so on.

For example, enter:

```
A> ASMFILE A:MYFILE B:MYPROG
```

AMSFILE is substituted for %0; A:MYFILE, for %1; and B:MYPROG, for %2.

The result is the same as if you had entered each of the commands (in MYFILE.BAT) from the console with the following parmeters:

```
REM START TEST BATCH FILE
COPY A:MYFILE.ASM B:MYPROG.ASM
TYPE B:MYPROG.PRN
TYPE A:ASMFILE.BAT
```

Remember that the dummy parameter %0 is always replaced by the drive specifier, if used, and the file name of the batch file.

NOTE:

- A maximum of 10 dummy parameters (%0 through %9) may be specified.

- To use % in a filename within a batch file, specify the % twice. For example, to specify the file ABC%.EXE, enter the name as ABC%%. EXE in the batch file.

CHKDSK (CHECK DISK) Command

Type: External.

Purpose: To scan the directory of the default or designated drive and check it for consistency.

Syntax: CHKDSK [*d*:]

Comments: Run CHKDSK occasionally on each diskette to verify the integrity of the directory structure. If any errors are found, the appropriate error message is displayed and corrective action can be attempted.

After the diskette has been checked, CHKDSK displays error messages, if any, and then a status report.

A sample status report follows:

```
322560  bytes total disk space
  9216  bytes in 2 hidden files
311296  bytes in 8 user files
  2048  bytes available on disk

131072  bytes total memory
118688  bytes free
```

CHKDSK assumes that the diskette being checked is already installed. On single-drive systems, it is important that the specified drive is different from the default (unless checking the DOS diskette itself).

If an error is detected, CHKDSK returns one of the following error messages:

Allocation error for file filename

The named file has a data block allocated to it that does not exist (that is, a data block number larger than the largest possible block number). CHKDSK truncates the file short of the bad block.

Disk not initialized

No directory or file allocation table is found. If files exist on the diskette, and the diskette has been damaged, it may still be possible to transfer files to another diskette to recover data.

Directory error-file: filename

No valid data blocks are allocated to the named file. CHKDSK deletes the file.

Files cross-linked: filename and filename

The same data block is allocated to both files. No corrective action is taken. To correct the problem, first use the COPYcommand to make copies of both files; then, delete the originals. Review each file for validity and edit as necessary.

File size error for file filename

The size of the file in a directory isdifferent from its actual size. The size in the directory is automatically adjusted to indicate its actual size on the diskette. The amount of useful data may be less than the size shown because the last data block may not be used fully.

XXXXXX bytes of disk space freed

Diskette space shown as allocated was not actually allocated and has been freed.

COMP (COMPARE FILES) Command

Type: External.

Purpose: To compare the contents of one file to those of another.

NOTE: Use COMP to compare individual files; use DISKCOMP to compare whole diskettes.

Syntax: COMP [*d*:] [*filename*] [*d*:] [*filename*]

Comments: The files being compared may exist on the same or different diskettes. The file names are optional. If they are omitted, DOS prompts you for the names and appropriate diskettes. The computer waits for you to press a key before it begins the comparison.

Since COMP compares the files byte-for-byte, any unequal bytes result in error messages. Additionally, the number of bytes that were compared are listed.

An error message might look like this:

```
Compare error at offset 128
File 1 = 32
File 2 = 20
```

After 10 unequal comparisons, COMP aborts, processing ends, and the following message is displayed:

```
10 Mismatches - aborting compare
```

When sucessfully compared, the screen displays:

```
Files compare ok
Compare more files? (Y/N)
```

This permits you to do additional comparisons.

NOTE:

- When the first file name is the same as the second file name, the specified drives must be different.

- Use of the wild card characters ? and * in the file name do not cause multiple file comparisons. Only the first file matching each name is compared.

- The computer does not make a comparison if the file sizes are different. The computer does ask you if you want to compare more files.

COPY Command

Type: Internal.

Purpose: To copy a file to the same or another diskette. When copying to a different diskette, the copied file may be given a different name, if you prefer. Files being copied on the same diskette **must** have different names.

Concatenation (combining files) may be performed during the COPY process.

Syntax: COPY [/A][/B] *source filename* [/A][/B][*d:*] *destination filename* [/A][/B][/V]

or

COPY [/A][/B] *source filename*[/A][/B][+*source filename* [/A][/B] [*d:*] [*destination filename*]] [/A][/B][/V]

Comments: **Verify Copy:**

The /V parameter causes DOS to verify the copy. Although errors in recording data are rare, this option has been provided for transferring critical data.

ASCII/Binary Copy:

Parameters /A and /B indicate the amount of data to be processed. Each applies to the file names preced-

ing it, and to all remaining file names on the command line until another /A or /B is encountered. These parameters can have the following meanings:

WHEN USED WITH A SOURCE FILENAME:

/A causes the file to be treated as a text (ASCII) file. The file's data is copied up to, but not including, the first end-of-file (EOF) character found in the file; the balance of the file is not copied.

/B causes the entire file to be copied.

WHEN USED WITH A DESTINATION FILENAME:

/A causes an EOF marker (^Z) to be added as the last character of the file.

/B causes no EOF marker to be added.

/A is a default value when concatenation is being performed (see Option 3, following). /B is the default when concatenation is not being performed (see Options 1 and 2).

Wild Card Characters:

The wild card characters ? and * may be used in file name parameters for both the source and destination filenames. When using a ? or * in the source filename, the names of the files are displayed as the files are copied.

The COPY command has three format options:

OPTION 1, The Source and Destination File Names Are the Same:

This option is useful when the destination file is to have the same name as the source file.

Example:

COPY *filename d*:

In this example, the file is copied to another diskette in the the specified drive. The destination file name is the same as the source, because no new name was specified. The computer will not copy a file to the same diskette without a new name.

OPTION 2, The Source and Target File Names Differ:

This option is used when the copied file is to have a different name from the source file. For example:

COPY *source copied*

or

COPY *source d*: *copied*

In the first example, the source file was copied to create the destination file on the same diskette. Because the names of both files are different, the computer assumes the drive to be used is the default drive.

Example:

COPY MYFCST.'82 B:*.JAN

This example demonstrates how to copy the file MYFCST.'82 from the default drive, Drive A:, to the diskette in Drive B:, and rename the copied file MYFCST.JAN.

Reserved device names may be used for the copy operation. Some examples are:

```
COPY CON: FILEA
COPY CON: AUX:
COPY CON: LPT1:
COPY FILEA CON:
COPY FILEB AUX:
COPY FILEC LPT1:
COPY AUX: CON:
COPY AUX: LPT1
```

NUL: may also be used as above.

Option 3, Concatenating (Combining) Files:

You may combine two or more files into one file by adding the additional files to the end of the first. The date and time recorded in the result file are the current date and time from when you combined the files.

Concatenation of files is accomplished by listing number of source files, separated by plus signs, in the COPY command, as shown in the following format:

COPY [/A][/B] *filename* [/A][/B] [+*filename* [/A][/B]] [*d*:] [*filename*] [/A][/B][/V]

Example:

```
COPY AFILE.XYZ+BFILE+B:CFILE.TXT ONEBIG.FIL
```

The COPY command now creates the new file, ONEBIG.FIL, on the diskette in the default drive. AFILE.XYZ, BFILE, and CFILE.TXT (on Drive B:) have now been added to each other and deposited into the ONEBIG.FIL.

If you don't specify a destination file name, the additional files are added to the end of the first file.

Concatenation is normally performed in text (ASCII) format. To combine binary files, use the /B parameter. This forces the COPY utility to use the physical EOF. You may combine ASCII and binary files through the use of either the /A (ASCII format) or /B (binary format) parameters.

Example:

```
COPY A.EXT+B.EXT/B+B:C.EXT/A BIGFILE
```

An /A or /B parameter takes effect on the file it is placed after, and applies to all subsequent files on the command line, or until another /A or /B is found.

You may use the wild card characters ? and * in the file names of both the source and destination files.

Example:

```
COPY *.LST RESULT.AAA
```

The result of this example is that all files matching *.LST are combined into one file called RESULT.AAA.

To enlarge on this concept, the following example combines all files with the extension of .MAN and .$$$ into one file called LABOR.FST.

```
COPY *.MAN+*.$$$ LABOR.FCST
```

If you attempt to copy a file to the same diskette without a new name, DOS detects an error and aborts the COPY command. When you are concatenating, as each source file is found, its name is compared to the name of the destination file. If the name is the same, the file is skipped, and the message "Content of destination lost before copy" is displayed. Further concatenation proceeds normally.

To avoid this complication, use the following command:

```
COPY REF.LST + *.LST
```

The result is that all *.LST files, except REF.LST, are appended to the file REF.LST .

SPECIAL CONSIDERATIONS OF CONCATENATION:

You must remember to include the /B parameter whenever the + sign is used with nontext (non-ASCII) files.

Example:

```
COPY B:FORECAST.'83+
```

In this example, the system copies the file, FORECAST.'83, located on the diskette in Drive B:, to the default diskette, giving it a new date and time.

When you are concatenating, the computer considers the copying process to be successful if at least one, but not necessarily all, of the named source files are found. If no source files are found, you will receive the message:

```
0 File(s) copied
```

CHANGING THE DATE OF A FILE NAME:

To alter the date and time during a copy procedure, use a command like the following:

```
COPY B:MYFILE.AAA+,, B:
```

NOTE: The two commas are necessary to define the end of the source filename because, in the COPY utility, the computer expects to see another file name after the plus sign.

DATE Command

Type: Internal.

Purpose: To display and set the date.

Syntax: DATE [*mm-dd-yy*]

Comments: If entered without a parameter, DATE returns with the message:

```
Current date is mm-dd-yy
Enter new date:
```

Press ENTER if you do not want to change the date shown.

Optionally, the date may be given as a parameter to the DATE command, as in:

```
DATE 3-9-81
```

In this case, no message appears.

The new date must be entered using numerals only; letters are not permitted. The allowable parameters are:

mm = 1-12
dd = 1-31
yy = 80-99 or 1980-2099

The date, month, and year entries may be separated by hyphens (-) or slashes (/). DOS is programmed to change months and years correctly, whether the month has 31, 30, 29, or 28 days. DOS handles leap years, too.

If the parameters or separators are not legal, DOS returns the message:

```
Invalid date
Enter new date:
```

DATE then waits for entry of a legal date.

DEL (DELETE) Command

Type: Internal.

Purpose: To delete specified files from a diskette and from the diskette directory.

Syntax: DEL *filespec*

Comments: If the *filespec* given for DEL is the global *.*, the prompt "Are you sure?" appears. If Y is typed as a response, then all files are deleted as requested. The ERASE command is the same as the DEL command.

DIR (DIRECTORY) Command

Type: Internal.

Purpose: To list a directory of the files on a diskette.

Syntax: [*d*:]DIR [*filespec*][/P][/W]

Comments: If no parameter is present, all directory entries on the diskette in the default drive are listed.

If only the drive specification is present (DIR *d*:), all entries on the diskette in the specified drive are listed.

If only a filename is present with no extension, all files with that filename on the diskette in the default drive are listed.

If a full file specification is present, all files with the specified filename and extension on the diskette in the specified drive are listed.

Wild Card Characters:

The wild card characters ? and * may be used in the *filespec* parameter. The following invocations of the DIR command are equivalent:

```
DIR          DIR *.*
DIR FILE     DIR FILE.*
DIR .EXT     DIR *.EXT
DIR .        DIR *.
```

Pause (/P) Parameter:

This parameter causes the display to pause when the screen is full. When you are ready to continue with the directory listing, press any key.

Wide (/W) Parameter:

This parameter produces a wide display of the directory. This display lists only the filenames. Each line displayed contains five filenames. It is recommended that this parameter be used on an 80-column display.

To List All Files:

Use this option to list all files in a directory.

Example:

DIR

or

DIR *d*:

Result:

```
A>DIR
FORECAST A      100      11-17-83     9:54p
FCST A          1099     12-01-82     3:00a
LABOR           20123     2-28-83    12:13p
```

To List Selected Files:

Use of this option permits you to list selected files in a directory.

Example:

DIR *filename*

or

DIR *d:filename*

If either filename or its extension is omitted, an * is assumed. The object of the first example is to list all files in the directory of the default drive. The second example illustrates how to list all files in the directory of a specified drive.

DISKCOMP (DISKETTE COMPARE) Command

Type: External.

Purpose: To compare the contents of one diskette to the contents of another diskette.

Syntax: DISKCOMP [*d:*] [*d:*] [/1]

Comments: Single or dual drives may be specified with this command. If you specify the same drive, a single-drive comparison is performed. The DISKCOMP utility prompts you as needed in the single-drive comparison.

Omitting both drive parameters causes a single-drive comparison to be performed on the diskette in the default drive. Specifying only one drive causes only the diskette in the specified drive to be compared.

If DISKCOMP is entered, and the COMPAQ Computer has two disk drives, DOS will prompt:

```
Insert first diskette in drive A:
Insert second diskette in drive B:
Strike any key when ready
```

If the appropriate diskettes have already been inserted in the appropriate disk drives, press any key to begin the comparison. As much data as can be stored in memory is taken from the first diskette and compared to the second diskette. Then the system goes back to the first diskette for more data. This process

continues until all the data on the first diskette has been compared to the data on the second diskette. DOS prompts:

```
Diskettes compare ok
Compare more diskettes? (Y/N)
```

Press Y to compare another pair of diskettes, or N to end the compare session.

If your COMPAQ Computer has one disk drive, DOS will prompt:

```
Insert first diskette in drive A:
Strike any key when ready
```

Insert the diskette, press any key, and the comparison begins. As much data as can be accommodated will be taken into memory from the diskette, then DOS prompts:

```
Insert second diskette in drive A:
Strike any key when ready
```

When you have exchanged diskettes and pressed a key, the data in memory will be compared with the data on the second diskette, and DOS will prompt for a return of the first diskette. This process continues, with alternation of the two diskettes in one drive, until the comparison is complete. DOS will display the same message as for a dual-drive system.

The /1 parameter forces DISKCOMP to compare only the first side of the diskettes, even if the diskettes and drives are double-sided.

DISKCOMP compares all 40 tracks on a track-for-track basis and issues a message if the tracks are not

the same. The message indicates the track number (0-39) and the side (0 or 1) where a mismatch was found.

If both drive specifiers are omitted, a single-drive comparison is performed on the default drive.

If the second drive specifier is omitted, the default drive is used as the secondary drive. If the default drive is specified in the first parameter, a single-drive comparison is made.

DISKCOMP may not issue a "Diskettes compare ok" message when a diskette created by the COPY command is compared to the diskette it was copied from. COPY produces a copy that contains the same information, but places the information at different locations on the target diskette from the locations on the source diskette. Use the COMP command to compare individual files on the diskettes.

If a diskette error occurs while DISKCOMP is reading the diskette, a message indicates where (track and side) the error occurred. Then DISKCOMP continues to compare the rest of the diskette. DISKCOMP automatically determines the number of sides to be compared, based on the diskette that is to be read first. If the first diskette or drive can be read on only one side, or if the /1 parameter is used, only the first side is compared on both diskettes. If the first drive and diskette are double-sided, a two-sided comparison is made. An error message is produced if the second diskette is single-sided.

DISKCOPY (COPY DISKETTE) Command

Type: External.

Purpose: To copy the contents of one diskette to another.

Syntax: DISKCOPY [*d:*] [*d:*] [/1]

Comments: The use of the optional /1 parameter causes DISKCOPY to to copy only the first side of the diskette, regardless of the diskette or drive type.

You may specify the same or different drives. Should the drives be the same, a single-drive copy operation is performed. You are prompted to insert the correct diskette at the appropriate time. DISKCOPY will wait for you to press any key before continuing.

If DISKCOPY A: B: is entered, and the COMPAQ Computer has two disk drives, DOS will prompt:

```
Insert first diskette in drive A:
Insert second diskette in drive B:
Strike any key when ready
```

If the appropriate diskettes have already been inserted in the appropriate disk drives, press any key to begin the copy procedure. As much data as can be stored in memory is taken from the first diskette and copied to the second diskette. Then the system goes back to the first diskette for more data. This process continues until all the data on the first diskette has been copied to the data on the second diskette. DOS prompts:

```
Copy complete ok
Copy another? (Y/N)
```

DISKCOPY automatically determines the number of sides to copy, based on the source diskette. If only the first side of the source diskette can be read, then only the first side will be copied. Should the source drive/diskette be dual-sided, both sides will be copied, unless you override it with the /1 parameter. Should any incompatibilities exist, the system displays an error message.

Parameter Omission Considerations:

The omission of both parameters causes the utility to perform a single-drive copy operation on the default drive.

Omitting the second parameter causes the default drive to be used as the destination drive.

Should you omit the second parameter and specify the default drive as the source drive, a single-drive copy operation will be performed.

Once complete, the DISKCOPY prompts:

Copy another? (Y/N)

Diskette Fragmentation Problems:

Diskettes having a large amount of file creations and deletions tend to become fragmented. This fragmentation occurs because diskettes are not designed to allocate space sequentially. The first free sector found is the next sector allocated, regardless of its location on the diskette.

Fragmentation will often cause degraded performance because of excessive head movement involved in finding, reading, and/or writing a file.

It is highly desirable that you resort to the COPY command instead of DISKCOPY to eliminate fragmentation.

For example:

```
COPY A:*.* B:
```

This exercise copies all files from the diskette in Drive A: to the diskette in Drive B:.

ERASE Command

Type: Internal.

Purpose: To delete a specified file from a designated drive or delete the file from the default drive (if a drive is not specified).

Syntax: ERASE *d:filename*
or
DEL *d:filename*

Comments: The shortened form, DEL, is a valid substitution for ERASE.

The use of wild card characters ? and * in file names is permitted. Wild card characters must be used with caution, because multiple files may be erased with a single command.

To erase/delete all files on a diskette, enter:

ERASE [*d*:]*.*

When you specify the filename *.* to erase all files on a diskette, DOS issues the following to verify the deletion request:

Are you sure? (Y/N)

EXE2BIN Command

Type: External.

Purpose: To convert files from the .EXE format to the .COM format.

Syntax: EXE2BIN *filespec* [*d*:][*filename*][.*ext*]

Comments: The first parameter is the input file; if no extension is given, it defaults to .EXE.

The second parameter is the output file. If no specific drive is given, the drive of the input file is used. If no filename is given, the filename of the input file is used. If no extension is given, .BIN is used.

The input file must be in valid .EXE format produced by the linker. The resident, or actual code and data contained in the file, must be less than 64 Kbytes. There must be no stack segment.

Two kinds of conversions are possible, depending on whether CS:IP: is specified.

If CS:IP is **not** specified, a pure binary conversion is assumed. If segment fix-ups are necessary, the following prompt appears:

Fix-up needed - base segment (hex):

If you type a legal hexadecimal number and then press ENTER, execution continues.

If CS:IP **is** specified as 100H, it is assumed the file is to be run as a .COM file ORGed at 100H, and the first 100H of the file is to be deleted. No segment fix-ups

are allowed, because .COM files must be segment relocatable.

If CS:IP does not meet one of these criteria, or meets the .COM file criterion but has segment fix-ups, the following error message is displayed:

File cannot be converted

To produce standard .COM files you must both ORG the file at 100H and specify the first location as the start address. This is done using the END statement. For example:

```
ORG         100H
START:
.
.
.
END         START
```

FORMAT Command

Type: Internal.

Purpose: To initialize the designated or default diskette in a recording format acceptable to DOS. Additionally, FORMAT analyzes the entire diskette for any defective tracks and prepares the diskette to accept DOS files by initializing the directory, file allocation table, and system loader.

Syntax: FORMAT [*d*:][/S][/1]

Comments: All diskettes must be formatted by using either the FORMAT or DISKCOPY command **before** the diskette may be used by DOS.

Formatting System Files (/S): When you use the /S parameter, the operating system files are also copied from the default drive diskette to the new diskette in the following order:

IOSYS.COM
MSDOS.COM
COMMAND.COM

Single-Sided Formatting (/1):

When you specify /1 as a parameter, the destination diskette is formatted for single-sided use.

NOTES:

- IOSYS.COM and MSDOS.COM are hidden from operator access. As such, they do not appear in any directory searches, including the DIR command.

- Formatting destroys all existing data on a diskette.

- Defective tracks are marked as reserved during the format process to prevent the tracks from being allocated to a data file.

- FORMAT produces a status report, indicating the following:

  Total diskette space.

  Space marked as defective.

  Space currently allocated to files (for use with /S).

  Available space.

- FORMAT determines the destination drive type and formats the diskette accordingly. If the diskette may be successfully read and written on on side only, the diskette is formatted for single-sided use. The diskette may be used in either type of drive. If the target drive is dual-sided and you do not use the /1 parameter, the diskette is formatted for dual-sided use; it will not be usable in a single-sided drive.

Example: By issuing the following command, the Drive B: diskette will be formatted and the operating system file will also be copied:

```
A> FORMAT B:/S
```

The system issues the following message:

```
Insert new diskette for drive B:
and strike any key when ready
```

After you insert the appropriate diskette and press any key, the system issues the message:

```
Formatting . . .
```

Once formatting is complete, the system issues the following message:

```
Formatting . . . Format complete
System transferred

322560 bytes total disk space
13312  bytes used by system
309248 bytes available on disk

Format another? (Y/N)
```

MODE Command

Type: External.

Purpose: To set the operations mode on 1) a printer; 2) an external monitor display; 3) the asynchronous communications adapter; or 4) to redirect all parallel printer output to the asynchronous communications adapter.

NOTE: When the MODE command is used for 1, 3, or 4, the printer and asynchronous communications adapter intercept code is placed in low memory in the DOS communications area.

Syntax: MODE [LPT#:][*n*][,*m*][,T]

or

MODE COM*n*:*baud*[[,*parity*][,*databits*][,*stopbits*][,P]

or

MODE LPT#:=COM*n*:

Comments: If there is an invalid or missing *n* or *m* parameter, the operations mode for that parameter will not change.

The MODE command has four possible formats:

Format 1, Printer:

MODE LPT#:[*n*][,*m*]

\# = printer number (1, 2, or 3).
n = characters per line (80 or 132).
m = vertical spacing, lines per inch (6 or 8).

Example: MODE LPT1: 132,8

The defaults for a printer are 80 characters per line, and 6 lines per vertical inch. This command changes the operations mode for printer 1 to 132 characters per line, and 8 lines per vertical inch.

Format 2; External Monitor Display:

MODE [*n*][,*m*][,T]

n = characters per line (40 or 80).
m = shift display, right or left (R or L).
T = request for a test pattern (used to align display).

With this command, you can shift the display one (for a 40-character line) or two (for an 80-character line) characters right or left to make viewing easier. When you include a T in this MODE command, a prompt asks you if the display is now aligned correctly. If you answer with Y, the command ends. If you answer with N, the display shifts again, and the prompt is repeated.

Example:
MODE 40,L,T

This command changes the operations mode to 40 characters per line, and shifts the display one character to the left. Then the test pattern is displayed, and the prompt appears. You can now shift the display again, without reentering the command, by answering the prompt with N.

Format 3, Asynchronous Communications Adapter:

MODE COM*n*:*baud*[,*parity*][,*databits*][,
stopbits][,P]

n =	asynchronous communications adapter number (1 or 2).
baud =	baud rate (110, 150, 300, 600, 1200, 4800, or 9600). You may enter only the first two numbers of the baud rate.
parity =	none (N), odd (O), or even (E). The default is even (E).
databits =	7 or 8. The default is 7.
stopbits =	1 or 2 If baud = 110, the default is 2; otherwise, the default is 1.

These are called the protocol parameters, and are used to initialize the asynchronous communications adapter. You must always specify *baud* in the protocol. You can use the defaults for the other parameters by entering commas.

Examples:
MODE COM2:30, O,8,2

This sets the operations mode on asynchronous communications adapter 2 to 300 baud rate, odd parity, 8 databits, and 2 stopbits. An example with defaults looks like this:

```
MODE COM2:30,,,,P
```

In this command, the parity defaults to even, the databits defaults to 7, and the stopbits defaults to one.

You can specify P **only** in protocol for a serial interface printer. The P parameter causes timeout errors to be retried continuously. To stop this loop,

press ^BREAK. You can stop the timeout errors from being retried continuously by reinitializing the asynchronous communications adapter without the P parameter in the command.

Format 4, Printer Output to Asynchronous Communications Adapter:

MODE LPT#:=COM*n*:

= printer number (1,2, or 3).
n = asynchronous communications adapter number (1 or 2).

Examples: LPT1:=COM2:

This command redirects all output for printer 1 to asynchronous communications adapter 2.

NOTE:

- You must initialize the asynchronous communications adapter (see Format 3) before using MODE to redirect parallel printer output to a serial device. You must include the P parameter in the initialization command if the serial device is a printer.

- To stop redirection of output for the printer designated with #, use:

MODE LPT#:[*n*][,*m*]

PAUSE Command

Type: Internal.

Purpose: To suspend execution of a batch file.

Syntax: PAUSE [*comment*]

Comments: To allow time for changing diskettes or performing other actions during the execution of batch command files, enter the PAUSE command. PAUSE suspends execution until any key, except ^BREAK, is pressed.

The optional comment may be entered on the same line as PAUSE to provide the operator of the batch files with some meaningful message during the pause. For example, a message may be given to change the diskette in a designated drive. The comment is displayed before the "Strike any key" message.

When PAUSE is encountered, the system suspends activity and displays:

```
Strike a key when ready
```

When you press any key, except ^BREAK, execution of the batch file resumes. If ^BREAK is pressed, the system displays the prompt:

```
Abort batch job? (Y/N)
```

If Y is pressed in response to this prompt, execution of the remainder of the batch file is aborted, and control returns to the DOS command level.

If N is pressed, execution of the batch file continues. PAUSE can be used to break a batch file into pieces, ending the file at an intermediate point.

REM (REMARK) Command

Type: Internal.

Purpose: To display a remark during the execution of a batch file.

Syntax: REM [*comment*]

Comments: The remark (or *comment*) is entered on the same line as REM, separated by a blank space, tab, or comma.

When the system encounters REM in a batch file, it will not attempt to execute the remark.

The REM command has no other purpose.

REN (RENAME) Command

Type: Internal.

Purpose: To rename a file.

Syntax: REN *filespec filespec*

Comments: The first *filespec* must be given a drive designation unless it resides in the default drive. Any drive designation for the second *filespec* is ignored. The renamed file continues to reside on the same diskette.

Use of a wild card character causes a one-to-one correspondence between files specified in the first *filespec* and those specified in the second *filespec.*

Examples: REN *.LST *.PRN

Renames all files with an .LST extension to the same names with a .PRN extension.

REN B:ABODE ?D?B?

Renames the file ABODE on the diskette in Drive B: to ADOBE. The file remains on the diskette in Drive B:.

SYS (SYSTEM) Command

Purpose: To transfer DOS system files from a diskette containing the files to the diskette in the designated drive.

Syntax: SYS *d*:

Comments: SYS is normally used to place the operating system files, IOSYS.COM and MSDOS.SYS, on a newly-formatted diskette. A drive specifier (*d*:) is required for the target diskette. IOSYS.COM and MSDOS.SYS are hidden files. These files do not appear in the listing when using the DIR command.

COMMAND.COM is not transferred.

FORMAT /S formats a new diskette and installs the system files.

TIME Command

Type: Internal.

Purpose: To display and set the time.

Syntax: TIME [*hh*][:*mm*][:*ss*][.*cc*]

Comments: If the TIME command is entered without parameters, the following message is displayed:

```
Current time is hh:mm:ss.cc
Enter new time:
```

A new time may be typed, or ENTER may be pressed to bypass the query.

The new time must be entered using numerals only; letters are not allowed. The allowable parameters are:

hh = 00-24 (hours).
mm = 00-59 (minutes).
ss = 00-59 (seconds).
cc = 00-99 (hundredths of seconds).

The hour, minute, second, and hundredths of seconds entries must be separated by colons.

DOS uses whatever time is entered as the new time as long as the parameters and separators are legal. If the parameters or separators are not legal, DOS returns the message:

```
Invalid time
Enter new time:
```

DOS then waits for a legal entry for time.

TYPE Command

Type: Internal.

Purpose: To display the contents of a file.

Syntax: TYPE *filespec*

Comments: When you use TYPE to display the contents of a file, the symbols displayed are the input as the computer sees it. Files containing text are readable, but some files, such as program files, contain many unrecognizable characters.

NOTE:

- This command allows you to examine the file, but not modify it. Use EDLIN to change the contents.

- With the TYPE command, the contents of the file scroll up the screen very quickly. To temporarily suspend the display, press ^NUM LOCK. Press any character key to continue the display.

- To print the contents of the file, press ^PRTSC before you enter the TYPE command. The contents of the file will print as they are displayed. To stop the printing, press ^PRTSC again.

3-4. SUMMARY OF DOS COMMANDS

(batch) Execution of batch files (internal command).
[*d*:]*filename*[parameters]

CHKDSK Scans the directory of the default or designated drive and checks for consistency, (external command).

CHKDSK [*d*:]

COMP Compares files (external command).

COMP [*filename*] [*d*:][*filename*]

COPY Copies specified file(s) (internal command).

COPY [/A][/B]*filename*[/A][/B] [*d*:][*filename*][/A][/B] [/V]

or

COPY [/A][/B]*filename*[/A][/B][+*filename*][/A][/B][*d*:] [*filename*][/A][/B][/V]

DATE Displays and sets date (internal command).

DATE [*mm-dd-yy*]

DEL Deletes specified file(s). DEL is the same as ERASE (internal command).

DEL [*d*:][*filename*]

DIR Lists requested directory entries (internal command).

DIR [*d*:][*filename*][/P][/W]

DISKCOMP Compares diskettes (external command).

DISKCOMP [*d*:] [*d*:] [/1]

DISKCOPY Copies the contents of one diskette to another diskette(external command).

DISKCOPY [*d*:] [*d*:] [/1]

ERASE Deletes specified file(s). ERASE is the same as DEL (internal command).

ERASE [*d*:][*filename*]

EXE2BIN Converts .EXE files to .COM format.

EXE2BIN filename [*d*:][*filename*]

FORMAT Formats a diskette to receive DOS files (external command).

FORMAT [*d*:][*filename*][/S][/1]

MODE Sets mode of operation on a printer or display (external command).

MODE [LPT#:][*n*][,*m*][,T]

or

MODE [*n*],[*m*],[T]

or

MODE COM*n*:*baud*[,*parity*][,*databits*][,*stopbits*][,P]

or

MODE LPT#:=COM*n*

PAUSE Pauses for input in a batch file (internal command).

PAUSE [*comment*]

REM Displays a comment in a batch file (internal command).

REM [*comment*]

REN Renames first file as second file (internal command).

REN[AME] *filename* [*d*:]*filename*

SYS Transfers DOS system files from one diskette toanother (external command).

SYS *d*:

TIME Displays and sets the time (internal command).

TIME [*hh:mm:ss.cc*]

TYPE Displays the contents of a specified file (internal command).

TYPE *filename*

4-1. INTRODUCTION TO EDLIN

EDLIN is a text editor used to edit files one line at a time. Each line may be up to 253 characters long. The last character of each, the end of line character, is signaled when the ENTER key is pressed.

EDLIN is invoked by entering:

EDLIN *d:filespec.ext*

If *d:filespec.ext* is not given, this message appears:

Filename must be specified

and the DOS prompt is returned. If the file is to be created, it must be given a name in advance.

If the specified file does not exist, EDLIN creates the file and returns the message "New file". The asterisk (*) prompt, and the cursor, indicate that input may begin.

If the specified file exists, EDLIN loads the file into memory. If the whole file is loaded, EDLIN returns the message "End of input file" and an asterisk (*) prompt. If the file is larger than memory, EDLIN fills three-fourths of available memory with the first part of the file, then returns the asterisk (*) prompt, but not the "End of input file" message. The lines in the file may be called up individually by typing the line number, or listed using the LIST (L) command.

Line numbers are not present in saved text, but when a file is created or called up for edit, lines are numbered dynamically. Line numbers begin at 1 and are incremented by 1 through the end of the file.

When new lines are inserted between existing lines, all line numbers following the inserted text are automatically incremented by the number of lines inserted. When lines are deleted between existing lines, all line numbers following the deleted text are decremented

automatically by the number of lines deleted. Consequently, line numbers always run from 1 through n (the number of the last line in the file).

If the file is longer than memory, and you want to edit the part that is not in memory, you must first write out to the diskette some of the file that is in memory, and then append more lines into memory, using the APPEND command.

To end the edit session, use the END (E) command to save the edit, or the QUIT (Q) command to quit the session without saving the edit. Q is especially useful if serious errors have been made during the editing session.

4-2. EDLIN COMMANDS

EDLIN supports both intraline and interline commands. Intraline commands perform editing functions within a line. Interline commands perform editing functions between lines.

Intraline Commands

Lines are edited in EDLIN through the use of special editing keys, and function keys which allow you to perform many small, but frequently used, commands with the touch of a key or two. Only those keys which have specific applications in EDLIN are presented here.

ENTER acts as the return key on a typewriter. It accepts a line and moves the cursor down to the first position of the next line.

The SPACEBAR advances the cursor one space (left to right) each time it is pressed. If a character is spaced over, it is deleted.

BACKSPACE moves the cursor one space (right to left) each time it is pressed. If a character is backspaced over, it is deleted.

The SHIFT keys on both sides of the keyboard change upper-case letters to lower-case letters, or vice-versa. SHIFT is always required to print the symbols above keyboard numbers.

CAPS LOCK changes the case of all characters, except numbers and symbols. Press once to activate; again to deactivate. If characters are lower-case, CAPS LOCK makes them upper-case. If they are upper-case, CAPS LOCK makes them lower-case. Whether or not CAPS LOCK is engaged, use SHIFT to print the opposite case, or a symbol above a number.

TAB moves the cursor to the next tab stop. Tab stops are set at every eight characters.

NUM LOCK is a toggle key. When it is activated, the numbers in the number pad to the right of the keyboard may be printed. When it is not activated (default) or is deactivated, only the CURSOR-LEFT and CURSOR-RIGHT keys are operational. The CURSOR-LEFT key moves the cursor, deleting characters, from right to left. The CURSOR-RIGHT key prints the characters from the template to the buffer input line, one at a time, as does the special editing key F1 (see below). The CURSOR-UP, CURSOR-DOWN, HOME, END, PGUP, and PGDN keys have no function in EDLIN.

Multiple and Control (CTRL) Keys

In this book, and in many program descriptions, there is a special symbol used to indicate control characters. For example, if the letter T is to be used as a control character, it is written as ^T. The caret (^) superscript always means that you must first press and hold down the CTRL key, and then press the designated character key. The ^T character is read aloud as "Control T".

If two keys are given (for example, ^BREAK), press and hold the first key, press the second key, and release them both. When three keys (for example, ^ALT+DEL) are given, press and hold the first two (CTRL and ALT), press the third (DEL), and release all three.

^ALT+DEL resets the system to the DOS command level.

^BREAK aborts the current command.

BACKSPACE moves the cursor and deletes one character (right to left) each time it is pressed.

^PRTSC activates the printer. Lines are printed as they are entered, or as they are displayed on the screen with the TYPE command. Press the keys again to deactivate the printer.

SHIFT+PRTSC causes the printer to print a hard copy of everything displayed on the screen, but it is not engaged on a continuing basis.

^NUM LOCK suspends display of output to the terminal screen temporarily. This is especially useful if a large amount of material is being output to the screen, as when a very long directory is being displayed with the DIR command, or a long file is being displayed with the TYPE command. To continue, press any character key.

Key:	**Function:**
F1	Copies one character from the template to the input buffer line. F1 is the same as the CURSOR-RIGHT key in the number pad.
F2	Copies all characters, up to the character specified, from the template to the input buffer line.
F3	Copies all remaining characters from the template to the input buffer line
DEL	Does not copy (skips over) a character in the template line and does not print it to the input buffer line. To see results of DEL, press F3.
F4	Does not copy (skips over) characters in the template up to the character specified. To see the results of F4, press F3.
ESC	Voids the current input buffer line, leaving the template unchanged.
INS	Turns on the insert mode. Allows characters to be inserted into a line. Press ENTER to cancel.
F5	Makes the last input buffer line the new template for further editing.

NOTE: Keys F6 through F10 have no functions assigned for EDLIN.

F1 Key

Purpose: To copy one character from the template to the input buffer line.

Comments: At the beginning of the intraline edit, the cursor (indicated by the underline) is positioned at the beginning of the input buffer line.

Pressing the F1 key copies one character from the template (the top line) to the input buffer line (bottom line). Insert is automatically turned off if it was on. If ENTER is pressed before the complete line has been copied, the template will be truncated from the last character before the entry.

NOTE: The CURSOR-RIGHT key in the number pad has the same function.

Example:

```
1:*This is a sample file.
1:*_
```

F1 copies the first character (T) from the template to the second, or input buffer line:

```
    1:*This is a sample file
F1  1:*T_
```

Each time F1 is pressed, one more character appears:

```
F1  1:*Th_
F1  1:*Thi_
F1  1:*This_
```

F2 Key

Purpose: To copy all characters, up to a specified character, from the template to the input buffer line.

Comments: Pressing F2 copies all characters, up to, but not including, a specified character. The specified character is the next character typed. It is not copied or shown on the screen. Pressing F2 causes the cursor to move to the specified character where it next appears in the line.

Use of the SPACEBAR as the single character causes all characters up to a space to be copied. If the template does not contain the specified character, nothing is copied. F2 automatically turns insert off, if it is on.

NOTE: F2 will not print the last character of a line from the template to the input buffer line. Use F1 or F3 to complete the line. If ENTER is pressed before the buffer line is complete, the template will be truncated from the point of entry.

Example: To copy all characters up to the character P:

```
1:*This is a sample file
F2p  1:*This is a sam_
```

F3 Key

Purpose: To copy the remainder of the line from the template to the input buffer line.

Comments: Pressing F3 copies all characters from the template, or copies all those remaining from the last character edited. When F3 is pressed, the rest of the line appears on the buffer line, and the cursor is positioned after the last character on the line. Insert is automatically turned off if it is on.

Only F1 or F3 will print the last character of the template to the input buffer line. If ENTER is pressed before the last character of the template is copied, the template will be truncated from the point of entry.

Example:

```
      1:*This is a sample file
F3    1:*This is a sample file _
```

DEL (DELETE) Key

Purpose: To skip over (not copy) one character from the template to the buffer line.

Comments: Each time DEL is pressed, one character is skipped over (not copied) from the template. The action of DEL is the opposite of F1. When the edited buffer line is entered, those characters not copied from the template are effectively deleted.

Example:

```
      1:*This is a sample file
DEL   1:*_
```

The cursor does not move. To see how much of the line has been skipped over, press F3 to copy the rest of the template to the buffer line.

```
      1:*This is a sample file
DEL   1:*_
F3    1:*his is a sample file _
```

F4 Key

Purpose: To skip over (not copy) multiple characters, up to but not including a specified character, from the template to the buffer line.

Comments: Pressing F4 skips over all characters in the template up to a specified character. The specified character is the next character typed. It is not copied and not shown on the screen.

To see the results of F4, press F3 to copy the rest of the template to the buffer line.

If the template does not contain the specified character, nothing is skipped over. The action of F4 is opposite to F2.

Example: To skip over all characters up to the character P:

```
        1:*This is a sample file
F4p F3  1:*ple file _
```

ESC (ESCAPE) Key

Purpose: To quit input and cancel the input buffer line.

Comments: Pressing ESC cancels the input buffer line but leaves the template unchanged. ESC also prints a backslash (\), moves the cursor to the first position of the next line, and turns insert mode off if it was on.

Pressing F3 copies the template to the input buffer line just as it was before ESC was pressed.

Example:

```
        1:*This is a sample file
F4p F3  1:*ple file
ESC     1:*ple file\
           _
```

Press F3 to copy the original template to the input buffer line again.

```
F3  This is a sample file_
```

INS (INSERT) Key

Purpose: To insert characters into a line.

Comments: Pressing INS causes the system to enter the insert mode. Characters are inserted behind the character the cursor points to. Characters ahead of the cursor are moved to the right to make room for the insert. After the desired characters are inserted, INS is pressed again to end the insert mode.

NOTE: If the ENTER key is pressed before INS is disabled, the rest of the line will be truncated.

Example:

```
        1:*This is a sample file
F2p     1:*This is a sam_
```

Press INS to insert the three characters s, o, and n.

```
F2p             1:*This is a sam_
INS (s o n)     1:*This is a samson_
```

Press INS again to exit insert mode and press F3 to copy the rest of the template:

```
INS F3    1:*This is a samsonple file_
```

If the ENTER key had been pressed, rather than F3, the remainder of the template would have been truncated, and the input buffer line ended at the end of the insert:

```
1:*This is a samson
```

F5 Key

Purpose: To make the edited input buffer line the new template for additional editing.

Comments: Pressing F5 makes the edited input buffer line the new template to allow further editing. Pressing F5, rather than ENTER, at the end of the edited line, outputs an at sign (@) at the end of the line, and positions the cursor, without the EDLIN prompt (*), under the first character of the next line. The line number is not shown.

Example:

```
                1:*This is a sample file
F2p             1:*This is a sam_
INS (s o n)     1:*This is a samson_
F3              1:*This is a samsonple file
F5              1:*This is a samsonple file@
                  _
```

Additional editing can now be done using the new template. To delete s o n and return the input buffer line to its original form:

NOTE: # = space in this example.

```
                1:*This is a sampsonple file
                  _

F2 #            This

F2 #            This is

F2 #            This is a
F1              This is a (#)
```

F1	1:*This is a s
F1	1:*This is a sa
F1	1:*This is a sam
DEL (3 times)	1:*This is a sam
F3	1:*This is a sample file

Interline Commands

Interline commands perform editing functions on whole lines at a time. The interline commands are summarized in the following list, and are described in detail following the description of command parameters.

Parameters:

Interline commands, except END (E) and QUIT (Q), accept some parameters. The effect of a parameter depends on the command.

Parameter: **Description:**

line *line* indicates a line number in a file. It is entered by the operator. Line numbers must be separated from other line numbers by a comma, and from other parameters and the command by a space. *line* may be specified in one of four ways:

Number Any integer less than 65534. If a number larger than the largest existing line number is specified, *line* indicates the line after the last line number.

(Period) If a period is specified for line, then *line* indicates the current line number. The current line is the most recently edited line, and not necessarily the last line displayed. It is marked by an asterisk (*) between the line number and the first character.

Parameter:	**Description:**
#	(Pound sign) The # indicates the line after the last line number. Specifying # for *line* has the same effect as specifying a number larger than the last line number.
ENTER	Pressing the ENTER key, without any of the line specifiers above, directs EDLIN to use a default value appropriate to the command.
line,line	A range of lines includes the first and last specified *lines* and all lines in between. Line numbers are separated by a comma; for example, 1,20 means lines 1 through line 20. A single line number indicates the specified line only. A line number followed by a comma (for example, 5,) specifies all lines from the given line through the end of the file. A line number followed by a comma and a period (for example, 5,.) specifies all lines from the given line through the current line. If only the second line number is given, it must be preceded by a comma. This specifies all lines from the beginning of the file through line 20. For example: ,20 specifies lines 1 through 20. If the period is used to specify the current line, and the comma is used (.,20), all lines from the current line through line 20 are specified.
n	Number of lines.

Continued

Continued

Parameter:	**Description:**
?	(Question mark) The question mark directs EDLIN to ask the operator if the correct string has been found. The question mark is used only with the REPLACE and SEARCH commands. Before continuing, EDLIN waits for either Y or ENTER as a yes response. Any other key signals a no response.
string	Specifies a string or group of characters to be found, to be replaced, or to replace other text. *string* is used only with the SEARCH and REPLACE commands. Each *string* must be terminated by a ^Z or an ENTER. No spaces should be left between strings, or between a string and its command letter, unless spaces are part of the string (see SEARCH and REPLACE Commands).

APPEND (A) Command

Purpose: To append lines from input file to editing buffer.

Syntax: [*n*]A

Comments: The APPEND command is used for extremely large files that will not fit into memory all at one time. By writing part of the editing buffer to the output file (using the WRITE (W) command), room is made for appending lines to the file using the APPEND (A) command.

If A is typed without a parameter, lines are appended to that part of the file currently in memory until available memory is three-quarters full, or until there are no more lines to append.

If the parameter *n* is given,then *n* lines are appended to that part of the file currently in memory.

DELETE (D) Command

Purpose: To delete the specified line or line range.

Syntax: [*line*][,*line*]D

Comments: If only one line number is given, the specified line is deleted.

If the line number is preceded by a comma, the second line number is assumed, and all lines up to, and including the specified line are deleted.

If both lines are specified, separated by a comma, both lines and all lines between are deleted.

A period indicates the current line and, when used with the comma, indicates the starting or ending line of the deletion.

NOTE: Use the DELETE (D) command with extreme caution. Lines and line ranges are permanently deleted. Should this happen, use the QUIT (Q) command to abort the edit and begin again.

When lines have been deleted, all line numbers following the deleted line or lines are dynamically renumbered. The line immediately after the deleted line or line range becomes the current line, and has the same line number as the first deleted line.

Example:

```
1: This is a sample file
2: used to demonstrate dynamic line numbers.
3: See what happens when you
4: delete and insert
.
.
.
25: (The D and I commands)
```

```
26: (Use ^Z to exit insert mode)
27:*line numbers
```

To delete lines 5 through 24:

```
5,24 D
```

The result is:

```
1: This is a sample file
2: used to demonstrate dynamic line numbers.
3: See what happens when you
4: delete and insert
5:*(The D and I commands)
6: (Use  Z to exit insert mode)
7: line numbers
```

To delete line 6, enter:

```
6 D
```

The result is:

```
1: This is a sample file
2: used to demonstrate dynamic line numbers.
3: See what happens when you
4: delete and insert
5: (The D and I commands)
6:*line numbers
```

To delete a range of lines, beginning with a current line (3) and ending with line 5 enter:

```
.,5 D
```

The result is:

```
1: This is a sample file
2: used to demonstrate dynamic line numbers.
3:*line numbers
```

EDIT Command

Purpose: To call up lines for editing.

Syntax: [*line number*]

Comments: When a line number is entered, EDLIN displays the line number and the text on that line. This is the template. The line number is reprinted on the line below, but the line is empty. This is the new input buffer line, and it is ready for editing.

Both lines show an asterisk (*) after the line number. The asterisk indicates that the line is the current line. In a listing, the current line is the last line entered.

Any of the available intraline commands can be used to edit the line. The existing text serves as the template. When the line has been edited, or if no changes are needed, press the ENTER key to accept the line.

When ENTER is pressed, EDLIN alternates between presenting the prompt and cursor (*_) at the left margin of the screen, and the next line of the file. Each time *_ appears, EDLIN will accept a new command (LIST, END, and so on).

If the ENTER key is pressed while the cursor is in the middle of a line, the remainder of the line will be truncated.

NOTE: If the input buffer line needs additional editing, use special editing key F5 to make the input buffer line a new template. This may be repeated as often as necessary. When the ENTER key is pressed, the final input buffer line is sent into memory.

Example:

```
1: This is a sample file
2: used to demonstrate
3: the editing of line
4:* four.
```

To edit line 4, enter:

```
4
```

The contents of the line are displayed with an asterisk on the template line. The input buffer line also has an asterisk. The cursor marks the first character space on the input buffer line.

```
4:*four.
4:*__
```

END (E) Command

Purpose: To exit the editing session, save the edited file, and return to DOS command level.

Syntax: E

Comments: To save the edited file on diskette, type E in response to the EDLIN prompt (*_). The system will store the file and exit to the DOS command level.

When a file has been created using the INSERT (I) command, type ^Z on an empty line following the last line of the insert. Then type E at the EDLIN prompt.

NOTE: In insert mode, ^Z will cause the line it is typed on to be deleted. It should always be typed on an empty line or a line below the last line of the file.

The E command takes no parameters.

You must be sure that the diskette contains enough free space for the entire file to be written. If the diskette does not contain enough free space, the write is aborted and some or all of the edited file is lost.

After execution of the E command, control is returned to the DOS command level.

To exit the editing session without saving the edit, use the QUIT (Q) command. This may be important if you have made a serious editing error such as entering D (DELETE) rather than L (LIST) after a line range. If the END command is used, the lines will be permanently deleted. The QUIT command will void all editing, including the errors, and you may edit again.

INSERT (I) Command

Purpose: To create a new file, or insert line(s) of text immediately before a specified line.

Syntax: [*line*]I

Comments: To create a file in EDLIN, enter:

```
A>EDLIN filename
```

EDLIN will respond:

```
New file.
*_
```

Enter the INSERT command (*) and EDLIN will prepare a line for input:

```
I
  1*_
```

Every time ENTER is pressed, another line will be added:

```
1:*_
2:*_
3:*_
4:*_
```

To insert lines into an existing file above the line specified (for example, 4), enter 4 I. Lines will be inserted above line 4. When the insert is concluded, line 4 and all subsequent lines will be automatically renumbered. The original line 4 will be the current line, regardless of its new line number.

To append lines to the end of a file, enter the number after the last line number (n I) or # I. # signifies the line number after the last line number in the file.

Enter ^Z to terminate input.

NOTE: ^Z cancels and deletes the line where it is typed. Enter ^Z on an empty line after the last line in the file.

When the insert is finished and insert mode has been exited, the line which now immediately follows the inserted lines in an existing file, becomes the current line. All line numbers following the inserted section are incremented by the number of lines inserted.

If line is not specified for an existing line, the default is the current line number; lines are inserted immediately before the current line.

If line is an integer larger than the last line number, or if # is specified as line, the inserted lines are appended to the end of the file. In this case, the last line inserted becomes the current line.

Example:

```
1: This is a sample file
2: used to demonstrate dynamic line numbers.
3: See what happens when you
4: delete and insert
5: (The D and I commands)
6: (Use ^Z to exit insert mode)
7:*line numbers
```

To insert text before a specific line (not the current line), enter a line number (for example, 4) and I:

```
4 I
```

The result is:

```
4:*_
```

Enter the new text for lines 4 and 5:

```
4:*fool around with
5:*those very useful commands that
```

To end the insert, type:

```
6:*^Z
```

The LIST command (L) will display the file:

```
1: This is a sample file
2: used to demonstrate dynamic line numbers.
3: See what happens when you
4: fool around with
5: those very useful commands that
6:*delete and insert
7: (The D and I commands)
8: (Use ^Z to exit insert mode)
9: line numbers
```

To insert lines immediately before the current line, enter:

```
I
```

The result is:

```
6:*_
```

Insert the following text and terminate witha ^Z:

```
6:*perform the two major editing functions,
7:*^Z
```

The result is:

```
 1: This is a sample file
 2: used to demonstrate dynamic line numbers.
 3: See what happens when you
 4: fool around with
 5: those very useful commands that
 6: perform the two major editing functions,
 7:*delete and insert
 8: (The D and I commands)
 9: (Use ^Z to exit insert mode)
10: line numbers
```

To append new lines to the end of the file, enter:

```
11 I
```

This produces:

```
11:*_
```

Enter the following new lines:

```
11:*The insert command can place new lines
12:*anywhere in the file; no space problems.
13:*Because the line numbers are dynamic,
14:*they'll slide all the way to 65533.
```

End insertion by typing ^Z. The new lines will appear at the end of all previous lines in the file. The result is:

The result is:

```
 1: This is a sample file
 2: used to demonstrate dynamic line numbers.
 3: See what happens when you
 4: fool around with
 5: those very useful commands that
 6: perform the two major editing functions,
 7: delete and insert
 8: (The D and I commands)
 9: (Use ^Z to exit insert mode)
10: line numbers
11: The insert command can place new lines
12: anywhere in the file; no space problems.
13: Because the line numbers are dynamic,
14: they'll slide all the way to 65533.
```

LIST (L) Command

Purpose: To display a file or a specified range of lines on the terminal screen.

Syntax: [*line*][,*line*]L

Comments: If L is entered without parameters, 23 lines are listed with the current line in the middle; that is, 11 lines before the current line, the current line, and 11 lines after the current line. The current line remains unchanged.

If both line numbers are given, separated by a comma, both lines and all lines between them are listed.

If only one line number is given, only that line is displayed.

If the line number is preceded by a comma, the 23 lines preceding and including specified line are displayed.

If the line number is followed by a comma, the specified line and the 23 lines following are displayed.

If a period is used to indicate the current line, the current line will be used as the starting or ending line of the display.

Example:

```
1: This is a sample file
2: used to demonstrate dynamic line numbers.
3: See what happens when you
4: delete and insert
5: (The D and I commands)
```

```
.
.
.
15:*The current line contains an asterisk.
.
.
.
26: (Use ^Z to exit insert mode)
27: line numbers
```

To list lines 2 through 5, enter:

```
2,5 L
```

The result is:

```
2: used to demonstrate dynamic line numbers.
3: See what happens when you
4: delete and insert
5: (The D and I commands)
```

To list a range of lines beginning with the current line (15), enter, line L:

```
.,26 L
```

The result is:

```
15:*The line contains an asterisk.
   .
   .
   .
26: (Use ^Z to exit insert mode)
```

To list a range of 23 lines beginning with a specifiedline (13), enter:

```
13, L
```

The result is:

```
13: The specified line is listed first in the range.
14: The line is unchanged by the L command.
15:*The line contains an asterisk.
.
.
.
35: ^Z exits interline insert command mode.
```

To list a range of 23 lines centered around the current line, enter only L:

```
L
```

The result is:

```
4: delete and insert
5: (The D and I commands)
.
.
.
13: The line is listed in the middle of the range.
14: The line remains unchanged by the L com-
    mand.
15:*The line contains an asterisk.
.
.
.
26: (Use ^Z to exit insert mode.)
```

QUIT (Q) Command

Purpose: To quit the editing session without saving editing changes, and exit to DOS command level.

Syntax: Q

Comments: The Q command takes no parameters. It is simply a fast means of exiting an editing session without saving the edit. QUIT is especially useful if severe errors were made in the edit, such as typing D (DELETE), rather than L (LIST) after a line range. Q will abort the edit, mistakes and all.

When the editing session is finished, and the EDLIN prompt (*_) appears, type Q.

As soon as the QUIT command is given, EDLIN displays the message:

```
Abort edit ? (Y/N)_
```

Press Y to quit the editing session; press N (or any character key) to continue the editing session.

To exit the editing session and save the edit, use the END (E) command.

REPLACE (R) Command

Purpose: To replace all occurrences of the first string in the specified range of lines with the second string.

Syntax: [*line*][,*line*] [?] R *string1* ^Z *string2*

Comment: As each occurrence of *string1* is found, it is replaced by *string2*. Each line in which a replacement occurs is displayed. If a line contains two or more replacements of *string1* with *string2*, the line is displayed once for each occurrence. When all occurrences of *string1* in the specified range are replaced by *string2*, the R command terminates and the asterisk prompt reappears.

If the question mark (?) parameter is given, the REPLACE command stops at each line with a string that matches *string1*, displays the line with *string2* in place, and then displays the prompt "O.K.?". If you press Y or the ENTER key, then *string2* replaces *string1*, and the next occurrence of *string1* is found. Again, the "O.K.?" prompt is displayed. This process continues until the end of the range or until the end of the file.

After the last occurrence of *string1* is found, EDLIN returns the asterisk prompt.

If any key besides Y or ENTER is pressed after the "O.K.?" prompt, *string1* is left as it was in the line, and REPLACE goes to the next occurrence of *string1*. Only the desired *string1* is replaced, and the replacement of embedded strings is prevented.

If a second string is to be given as a replacement, then *string1* must be terminated with a ^Z.

If the *string2* is omitted, then *string1* may be terminated with either ^Z+ENTER or simply ENTER. *string2* must also be terminated with ^Z+ENTER or with a simple ENTER.

If *string1* is omitted, the replacement is terminated immediately.

If *string2* is omitted, *string1* is deleted from all lines in the range.

If the first line is omitted in the range argument (as in ,*line*) then the first line defaults to the line after the current line.

If the second line is omitted (as in *line* or *line*,) the second line defaults to #. # indicates the line after the last line of the file. This is the same as *line*,#.

Example:

```
1: This is a sample file
2: used to demonstrate dynamic line numbers.
3: See what happens when you
4: fool around with
5: those very useful commands that
6: perform the two major editing functions,
7: delete and insert
8: (The D and I commands)
9: (Use ^Z to exit insert mode)
10: line numbers
11: The insert command can place new lines
12: anywhere in the file; no space problems.
13: Because the line numbers are dynamic,
14: they'll slide all the way to 65533.
```

To replace all occurrences of *string1* (and) with *string2* (or) in a specified range, enter:

```
2,12 Rand ^Z or ENTER
```

The result is:

```
5: those very useful commors that
7: delete or insert
8: (The D or I commands)
8: (The D or I commors)
11: The insert commor can place new lines
```

In the replacement, some unwanted substitutions have occurred. To avoid these, and confirm each replacement, the same original file can be used with a slightly different command. To replace only certain occurrences of the first string with the second string, enter:

```
2? Rand (^Z or ENTER)
```

The result is:

```
5: those very useful commors that
O.K.? N
7: delete or insert
O.K.? Y
8: (The D or I commands)
O.K.? Y
8: (The D or I commors)
O.K.? N
11: The insert commor can place new lines
O.K.? N
*_
```

The LIST (L) command can be used to see the result of all these changes:

```
.
.
.
5: those very useful commands that
.
7: delete or insert
8: (The D or I commands)
.
11: The insert command can place new lines
.
.
.
```

SEARCH (S) Command

Purpose: To search the specified range of lines for the specified *string*.

Syntax: [*line*][,*line*] [?] S*string*

Comments: The *string* must be terminated with an ENTER. The first line that matches *string* is displayed and becomes the current line. The SEARCH command terminates when a match is found. If no line contains a match for *string*, the message "Not found" is displayed.

If the optional parameter, question mark (?), is included in the command, EDLIN displays the first line with a matching *string*; it then prompts you with the message "O.K.?". If you press either Y or ENTER, the line becomes the current line and the search terminates. If you press any other key, the search continues until another match is found, or until all lines have been searched; then the "Not found" message is displayed.

If the first line is omitted (as in ,*line* S*string*), the first line defaults to the line after the current line. If the second line is omitted (as in *line* S*string* or *line*, S*string*), the second line defaults to #, which is the same as *line*,# S*string*. If *string* is omitted, no search is made and the command terminates immediately.

Example:

```
1: This is a sample file
2: used to demonstrate dynamic line numbers.
3: See what happens when you
4: fool around with
5: those very useful commands that
```

```
6: perform the two major editing functions,
7: delete and insert
8: (The D and I commands)
9: (Use ^Z to exit insert mode)
10: line numbers
11: The insert command can place new lines
12: anywhere in the file; no space problems.
13: Because the line numbers are dynamic,
14:*they'll slide all the way to 65533.
```

To search for the first occurrence of the *string* (and),enter:

```
2,12 Sand (ENTER)
```

The result is:

```
5:*those very useful commands that
```

To get to the and in line 7, modify the search command by entering:

```
DEL F3,12 Sand (ENTER)
```

The search begins from the line after the current line (line 5), since no first line is given. The result is:

```
7: delete and insert
```

To search through several occurrences of a string until the correct string is found, enter:

```
1, ? Sand (ENTER)
```

The result is:

```
5:*those very useful commands that
O.K.?_
```

If any key except Y or ENTER is pressed, the search continues. Type N to continue.

```
O.K.? N

Continue:

7:*delete and insert
O.K.?_

Press Y to terminate the search:

O.K.? Y
*_
```

WRITE (W) Command

Purpose: To write lines from the editing buffer to the output file.

Syntax: [*n*]W

Comments: The WRITE command is used when editing files that are larger than available memory. By executing WRITE, lines are written out to the output file, and room is made in the editing buffer for more lines to be appended from the input file.

If W is typed with no *n* (number of lines) parameter, lines are written out until memory is one-quarter full.

If the *n* parameter is given, *n* lines are written out. Lines are written out beginning with the start of the file (line 1). Subsequent lines in the editing buffer are renumbered beginning with 1. A later APPEND command will append lines to any remaining lines in the editing buffer.

4-3. EDLIN ERROR MESSAGES

EDLIN error messages occur either when EDLIN is invoked, or during the actual editing session.

Errors When Invoking EDLIN

Cannot edit .BAK file—rename file

Cause: You have attempted to edit a file with the filename extension .BAK. .BAK files cannot be edited because the extension is reserved for back-up copies.

Cure: If you need the .BAK file for editing purposes, you must either rename the file with a different extension, or copy the .BAK file with a different filename extension.

No room in directory for file

Cause: When you attempted to create a new file, either the file directory was full, or you specified an illegal disk drive or an illegal filename.

Cure: Check the EDLIN invocation command line for illegal filename or illegal disk drive entries. If the command is no longer on the screen, and if you have not yet entered a new command, the EDLIN invocation command can be recovered by pressing the F3 key.

If the invocation command line contains no illegal entries, run the CHKDSK program for the specified diskette. If the status report shows the diskette directory full, remove the diskette and insert and format a new diskette. If the CHKDSK status report shows the diskette directory is not full, check the EDLIN invocation command for an illegal filename or illegal disk drive designation.

Errors While Editing

Entry Error

Cause: The last command entered contained a syntax error.

Cure: Reenter the command with the correct syntax.

Line too long

Cause: During REPLACE command mode, the *string* given as the replacement causes the line to expand beyond the limit of 254 characters. EDLIN aborts the REPLACE command.

Cure: Divide the long line into two lines, then retry the REPLACE command.

Disk Full—file write not completed

Cause: You gave the END command, but the diskette did not contain enough free space for the whole file. EDLIN aborts the E command and returns to the operating system. Some of the file may have been written to the diskette.

Cure: Only a portion, if any, of the file will have been saved. You should probably delete whatever file was saved and restart the editing session. None of the file which was not written out will be available after this error. Always be sure that the diskette has sufficient free space for the file to be written before beginning the editing session.

5-1. INTRODUCTION TO DEBUG

DEBUG is a debugging program used to provide a controlled testing environment for binary and executable object files. DEBUG eliminates the need to reassemble a program to see if a problem has been fixed by a minor change. It permits the alteration of file or CPU register contents. Once altered, DEBUG can immediately reexecute a program to check the validity of the changes.

All DEBUG commands may be aborted at any time by pressing ^BREAK. ^NUM LOCK suspends the display, so that you may read it before it scrolls away. Pressing any key other than ^BREAK or ^NUM LOCK restarts the display.

5-2. STARTING DEBUG

To start the DEBUG program, enter:

DEBUG [*d:*][*filespec*][*parameter*][*arglist*]

For example, if a *filespec* is specified, then the following is a typical invocation:

DEBUG COMP.COM

At start-up, the segment registers are set to the bottom of free memory, the instruction pointer is set to 0100H, the stack pointer is set to the end of the segment (or the bottom of the transient portion of COMMAND.COM), all flags are cleared, and the remaining registers are set to zero.

DEBUG loads COMP.COM into memory, starting at 100 hexadecimal in the lowest available segment. The BX:CX registers are loaded with the number of bytes placed in memory.

arglist may be specified if *filespec* is present. These filename parameters and switches are passed to the program *filespec*. When *filespec* is loaded into memory, it is loaded as if it had been invoked with the command:

filespec arglist

Here, *filespec* is the file to be debugged, and *arglist* is the rest of the command line.

If no *filespec* is specified, DEBUG is invoked as follows:

DEBUG

DEBUG returns with the prompt, — , signaling that it is ready to accept your commands. Since no filename has been specified, current memory, disk sectors, or disk files can be worked on by invoking later commands.

5-3. DEBUG COMMANDS

Each DEBUG command consists of a single letter followed by one or more parameters. Additionally, the control characters and the special editing function keys described in Chapter 4, all apply to DEBUG.

If a syntax error occurs in a DEBUG command, DEBUG reprints the command line and indicates the error with a caret (^) and the word ERROR.

For example:

```
DCS:100 CS:110
      ^ERROR
```

All commands and parameters may be entered in either upper and/or lower case.

The DEBUG commands are summarized below, and are described in detail following the description of command parameters.

DEBUG Commands Summary

Command:	**Syntax:**
C (COMPARE)	C *range address*
D (DUMP)	D [*address* [*L value*]] D [*range*]
E (ENTER)	E *address* [*list*]
F (FILL)	F *range list*
G (GO)	G [=*address* [*address . . .*]]

Continued

Continued

Command:	**Syntax:**
H (HEX)	H *value value*
I (INPUT)	I *address*
L (LOAD)	L [*address* [*d: sector sector*]]
M (MOVE)	M *range address*
N (NAME)	N *filename* [*filename . . .*]
O (OUTPUT)	O *address byte*
Q (QUIT)	Q
R (REGISTER)	R [*register name*]
S (SEARCH)	S *range list*
T (TRACE)	T [= *address*][*value*]
U (UNASSEMBLE)	U [*address* [*L value*]] U [*range*]
W (WRITE)	W [*address* [*d: sector sector*]]

Parameters

As shown in the summary, all DEBUG commands accept parameters, except the QUIT command. Parameters may be separated by delimiters (spaces or commas), but a delimiter is required only between two consecutive hexadecimal values. The following commands are equivalent:

```
DCS:100 110
D CS:100 110
D,CS:100,110
```

Parameter:	**Definition:**
d:	A 1-digit hexadecimal value indicating which drive a file is loaded from or written to. The valid values are 0 and 1. These values designate the drives as follows: 0=A: and 1=B:.
byte	A 1- or 2-digit hexadecimal value to be placed in, or read from, an address or register.
sector	A 1- to 3-digit hexadecimal value used to indicate the logical *record* number on the diskette and the number of diskette sectors to be written or loaded. Logical records correspond to sectors. However, the numbering differs since the entire diskette space is represented.
value	One to four hexadecimal *value* digits used to specify a port number or the number of times a command is to repeat its functions.
address	A 1- or 2-part designation which consists of either an alphabetic segment register designation, or a 4-digit segment *address*, plus an offset value. If the segment designation or segment *address* is omitted, the default segment is used. DS is the default segment for all commands except GO, LOAD, TRACE, UNASSEMBLE, and WRITE, for which the default segment is CS. All numeric values are hexadecimal. For example: CS:0100 04BA:0100 The colon is required between a segment designation (whether numeric or alphabetic) and an offset.
range	Enter either of the following formats to specify the upper and lower addresses of a range.

Continued

Parameter: **Definition:**

address address
or
address L value

Example:

```
CS:100 110
CS:100 L 10
```

The following is illegal:

```
CS:100 CS:110
       ^ERROR
```

The limit for *range* is 10000 hex.

list A series of byte values or of strings. *list* must be the last parameter on the command line. For example:

```
FCS:100 42 45 52 54 41
```

string Any number of characters enclosed in quotation marks. Quotation marks may be either single (') or double ("). Within *string*, the opposite set of quotation marks may be used freely as literals. If the delimiter quotation marks must appear within a *string*, the quotation marks must be doubled. For example:

```
'This ''string'' is valid.'
'This ' 'string' ' is valid.'

'This 'string' is not.'

"This 'string' is valid."
"This " "string" " is valid."

"This "string" is not."
```

Parameter: **Definition:**

Double quotation marks are not necessary in the following *strings*:

"This ' 'string' ' is not necessary."
'This " "string" " is not necessary.'

The ASCII values of the characters in the *string* are used as a list of byte values.

COMPARE (C) Command

Purpose: To compare the portion of memory specified by *range* to a portion of the same size beginning at *address*.

Syntax: C *range address*

Comments: If the two areas of memory are identical, there is no display, and DEBUG returns with the DOS prompt. If there are differences, they are displayed as:

address1 byte1 byte2 *address2*

Example: The following commands have the same effect:

```
C100,200 300
C100L100 300
```

Each command compares the block of memory from 100 to 200H, with the block of memory from 300 to 400H.

DUMP (D) Command

Purpose: To display the memory contents of either a single *address*, a *range* of *addresses*, or the number of lines specified by *value* beginning at the *address* specified.

Syntax: D[*address*[*L value*]]

D[*range*]

Comments: If a single *address* only is specified, the contents of 128 bytes are displayed.

If a *range* of *addresses* is specified, the contents of the *range* are displayed.

If the D command is entered without parameters, the result is the same as if you specify a single *address*, except that the display begins at the current location in the data segment (DS).

The dump is displayed in two portions: a hexadecimal dump (each byte is shown in hexadecimal value) and an ASCII dump (the bytes are shown in ASCII characters). Nonprinting characters are denoted by a period (.) in the ASCII portion of the display.

Each displayed line begins on a 16-byte boundary and the first line contains fewer bytes than the rest of the displayed lines.

NOTE: Each display line shows 16 bytes with a hyphen between the eighth and ninth bytes. When

the displays require more space than page width permits, they are divided into two lines. For example:

```
04B5:0100 42 45 52 54 41 20 54 00-20 42
  4F 52 4C 41 4E 44 BERTA T. BORLAND
```

Examples: The command:

```
dcs:100 10F
```

displays:

```
04BA:0100 42 45 52 54 41 . . . 4E 44 BERTA T. BORLAND
```

Enter:

```
D
```

The display is formatted as described above. Each line of the display begins with an address incremented by 16 bytes from the address on the previous line. Each subsequent D (entered without parameters) displays the bytes immediately following those last displayed.

If you enter the command:

```
DCS:100 L 20
```

the display is formatted as described above, with all bytes within the range of 100H to 120H in the CS segment displayed.

If you enter the command:

DCS:100 115

the display is formatted as described above, with all the bytes in the *range* from 100H to 115H in the CS segment displayed.

ENTER (E) Command

Purpose: To enter the *address*, display its contents, and wait for input.

Syntax: E *address* [*list*]

Comments: If the optional *list* of hexadecimal values is entered, the replacement of byte values occurs automatically. If an error occurs, no byte values are changed.

If the *address* is entered without the optional *list*, DEBUG displays the *address* and its contents, then repeats the *address* on the next line and waits for your input. The ENTER command performs one of the following actions:

1. Replace a byte value with a value you type in. You type the value after the current value. If the value typed in is not a legal hexadecimal value, or if more than two digits are typed, a beep sounds and the illegal or extra character is not echoed.

2. Press the SPACEBAR to advance to the next byte. To change the value, enter the new value as described above. If you space beyond an 8-byte boundary, DEBUG starts a new display line with the *address* displayed at the beginning.

3. Type a hyphen (-) to return to the preceding byte. If you decide to change a byte behind the current position, typing the hyphen returns the current position to the previous byte. When the hyphen is typed, a new line is started with the *address* and its byte value displayed.

4. Press the ENTER key to terminate the ENTER command. The ENTER key may be pressed at any byte position.

Examples: ECS:100

DEBUG displays:

```
04BA:0100 EB._
```

To change this value to 41, enter 41 as shown:

```
04BA:0100 EB.41_
```

To step through the subsequent bytes, press the SPACEBAR:

```
04BA:0100 EB.41  10.  00.  BC._
```

To change BC to 42:

```
04BA:0100 EB.41  10.  00.  BC.42_
```

To correct 10 to 6F, enter the hyphen as many times as needed to return to byte 0101 (value 10), then replace 10 with 6F:

```
04BA:0100 EB.41  10.  00.  BC.42—

04BA:0102 00.—_
04BA:0101 10.6F_
```

When you press the ENTER key, it ends the ENTER command and returns to the DEBUG command level.

FILL (F) Command

Purpose: To fill the addresses in *range* with the values in the *list*.

Syntax: F *range list*

Comments: If the *range* contains more bytes than the number of values in the *list*, the *list* is used repeatedly until all bytes in the *range* are filled.

If the *list* contains more values than the number of bytes in the *range*, the extra values in the *list* are ignored.

If any of the memory in the *range* is not valid (bad or nonexistent), the error is propagated into all succeeding locations.

The FILL command does not abort as the ENTER command does. The FILL command is a multiple version of the ENTER command that allows you to change more than one address at a time.

Example: F04BA:100 L 100 42 45 52 54 41

DEBUG fills memory locations 04BA:100 through 04BA:200 with the bytes specified. The five values are repeated until all 100H bytes are filled.

GO (G) Command

Purpose: To execute the program currently in memory.

Syntax: G [=*address* [*address* . . .]]

Comments: If the GO command is entered alone, the program executes as if it had run outside DEBUG.

If =*address* is set, execution begins at the address specified. If the segment designation is omitted from =*address*, only the instruction pointer is set. If the segment designation is included in =*address*, both the CS segment and the instruction pointer are set. The equals sign (=) is required so that DEBUG can distinguish the start =*address* from the breakpoint addresses.

When program execution reaches the *address* of a breakpoint, execution halts and control is returned to DEBUG. All registers, flags, and the decoded instruction are displayed for the last instruction executed. The result is the same as if you had entered the REGISTER command for the breakpoint *address*. Enter the GO command again to resume execution.

As many as 10 breakpoints may be set. Breakpoints may be set only at *addresses* containing the first byte of an 8086 opcode. If more than 10 breakpoints are set, DEBUG returns the BP error message.

The user stack pointer must be valid and must have six bytes available for this command.

The GO command uses an IRET instruction to cause a jump to the program under test. The user stack pointer is set, and the user flags, code segment register, and instruction pointer are pushed onto the user stack. If the user stack is not valid, or is too small, DOS may crash.

An interrupt code (0CCH) is placed at the specified breakpoint *address(es)*. When an instruction with the breakpoint code is encountered, all breakpoint *addresses* are restored to their original instructions. If execution is not halted at one of the breakpoints, the interrupt codes are not replaced with the original instructions.

Example: GCS:7550

The program currently in memory executes up to the address 7550 in the CS segment. DEBUG displays registers and flags, and the GO command is terminated.

After a breakpoint has been encounterd, if the GO command is entered again, and the program executes as if you had entered the filename at the DOS command level. The only difference is that program execution begins at the instruction after the breakpoint, rather than at the usual start *address*.

HEX (H) Command

Purpose: To perform hexadecimal arithmetic on two parameters.

Syntax: H *value value*

Comments: DEBUG adds the two parameters, then subtracts the second parameter from the first. The results of the arithmetic is displayed on one line; first the sum, then the difference.

Example: H19F 10A

DEBUG performs the calculations and then returns the results:

02A9 0095

INPUT (I) Command

Purpose: To input and display one byte from the port specified by *address*.

Syntax: I *address*

Comments: A 16-bit port address is allowed.

Example: I2F8

If the byte at the port is 42H, DEBUG inputs the byte and displays the *value*:

42

LOAD (L) Command

Purpose: To load a file into memory.

Syntax: L [*address* [*d: sector sector*]]

Comments: Set BX:CX to the number of bytes read. The file must have been named either with the DEBUG invocation command or with the NAME command. Either the invocation or the NAME command formats a file-name properly in the normal format of a file control block at CS:5C.

If the LOAD command is given without any parameters, DEBUG loads the file into memory beginning at address CS:100 and sets BX:CX to the number of bytes loaded.

For example:

```
A>DEBUG
—NFILE.COM
```

To load FILE.COM, enter:

```
L
```

If LOAD command is entered with an *address* parameter, loading begins at the memory *address* specified.

If LOAD is entered with all parameters, absolute diskette sectors are loaded, rather than a file. The *records* are taken from the drive specified. The drive specifier is numeric; for example, 0=A: and 1=B:.

DEBUG begins loading with the first *sector* specified, and continues until the number of sectors specified in the second *sector* have been loaded.

For example:

```
L04BA:100 1 0F 6D
```

DEBUG loads 109 (6D hex) *sectors*, beginning with logical *sector* number 15 (0F hex), into memory beginning at address 04BA:0100. When the sectors have been loaded, the DEBUG prompt returns.

If the file has a .EXE extension, it is relocated to the load *address* specified in the header of the .EXE file; the *address* parameter is always ignored for .EXE files. The header is stripped off the .EXE file before it is loaded into memory; the size of a .EXE file on diskette differs from its size in memory.

If the file named by the NAME command, or specified on invocation, is a .HEX file, entering LOAD with no parameters causes loading of the file to begin at the *address* specified in the .HEX file.

If the LOAD command includes the option *address*, DEBUG adds the *address* specified in LOAD to the *address* found in the .HEX file to determine the start *address* for loading the file.

MOVE (M) Command

Purpose: To move the block of memory specified by *range* to the location beginning at the specified *address*.

Syntax: M *range address*

Comments: Overlapping moves (moves where part of the block overlaps some of the current *addresses*) are always performed without loss of data. Addresses that could be overwritten are moved first.

To move from higher addresses to lower addresses, move data beginning at the block's lowest address and work towards the highest. To move from lower addresses to higher addresses, move data beginning at the block's highest address and work towards the lowest.

If the addresses in the block being moved will not have new data written to them, the existing data remains. The sequence of the move is important because the MOVE command copies the data from one area into another, in the sequence described, and writes over the new addresses.

Example: Enter:

```
MCS:100 110 CS:500
```

DEBUG first moves address CS:110 to address CS:510; then CS:10F to CS:50F, and so on until CS:100 is moved to CS:500. Enter the DUMP command, using the *address* entered for the MOVE command, to review the results of the move.

NAME (N) Command

Purpose: To set *filenames*.

Syntax: N *filename [filename . . .]*

Comments: The NAME command performs two distinct functions, both associated with *filenames*.

First, NAME is used to assign a *filename* for a later LOAD or WRITE command. If you invoke DEBUG without naming any file to be debugged, the NAME command must be given before a file can be loaded.

Second, NAME is used to assign *filename* parameters to the file being debugged. In this case, NAME accepts a list of parameters used by the file being debugged.

These functions overlap. Consider the following set of DEBUG commands:

```
—NFILE1.EXE
—L
—G
```

NAME assigns the filename FILE1.EXE to the *filename* to be used in any later LOAD or WRITE commands.

NAME also assigns the filename FILE.EXE to the first *filename* parameter to be used by any program that is later debugged.

LOAD loads FILE.EXE into memory.

GO causes FILE.EXE to be executed with FILE.EXE as the single *filename* parameter; for example, FILE.EXE is executed as if FILE FILE.EXE had been typed at the command level.

```
—NFILE1.EXE
—L
—NFILE2.DAT FILE3.DAT
—G
```

NAME sets FILE1.EXE as the *filename* for the subsequent LOAD command. The LOAD command loads FILE1.EXE into memory, and then the NAME command is used again, to specify the parameters to be used by FILE1.EXE.

When the GO command is executed, FILE1.EXE is executed as if FILE1 FILE2.DAT FILE3.DAT had been typed at the DOS command level.

If a WRITE command were executed at this point, FILE1.EXE, the file being debugged, would be saved with the name FILE2.DAT. To avoid such undesired results, always execute a NAME command before either LOAD or WRITE command.

Four regions of memory can be affected by the NAME command:

```
CS:5C  FCB for file 1.
CS:6C  FCB for file 2.
CS:80  Count of characters.
CS:81  All characters entered.
```

A file control block (FCB) for the first *filename* parameter given to the NAME command is set up at CS:5C. If a second *filename* parameter is given, an FCB is set up for it beginning at CS:6C. The number of

characters typed in the NAME command, exclusive of N, is given at location CS:80. The actual stream of characters given by the NAME command, exclusive of N, begins at CS:81. This stream of characters may contain any switches and delimiters that would be legal in any command typed at the DOS command level.

Example:

```
DEBUG PROG.COM
—NPARAM1 PARAM2/C
—G
```

The GO command executes the file in memory as if the following command line had been entered:

PROG PARAM1 PARAM2/C

Testing and debugging reflect a normal runtime environment for PROG.COM.

OUTPUT (O) Command

Purpose: To send the specified byte to the output port specified by *address*.

Syntax: O *address byte*

Comments: A 16-bit port address is allowed.

Example: Enter:

```
O2F8 4F
```

DEBUG outputs the byte value 4F to output port 2F8.

QUIT (Q) Command

Purpose: To terminate DEBUG.

Syntax: Q

Comments: The QUIT command takes no parameters, exits DEBUG without saving the file currently being operated on, and returns to the DOS commands level.

Example: Enter:

```
Q
```

DEBUG is terminated, and control returns to the DOS command level.

REGISTER (R) Command

Purpose: To display the contents of one or more CPU registers.

Syntax: R [*register name*]

Comments: If no *register name* is entered, the REGISTER command dumps the register-save area and displays the contents of all registers and flags.

If a *register name* is entered, the 16-bit value of that register is displayed in hexadecimal, and a colon appears as a prompt. Either enter a value to change the register, or press the ENTER key if no change is wanted.

The only valid *register names* are:

AX	BP	SS
BX	SI	CS
CX	DI	IP
DX	DS	PC
SP	ES	F

IP and PC both refer to the instruction pointer.

Any other entry for *register name* results in a BR error message.

If F is entered as the *register name*, DEBUG displays a two-character alphabetic code. To alter the flag, enter the opposite two-letter code. The flags are either set or cleared. The flags and their set and clear codes are listed in Table 5-1.

Table 5-1.
Set and Clear Codes for Flags

Flag Name:	Set:	Clear:
Overflow	OV	NV
Direction	DN decrement	UP increment
Interrupt	EI enabled	DI disabled
Sign	NG negative	PL plus
Zero	ZR	NZ
Auxiliary Carry	AC	NA
Parity	PE even	PO odd
Carry	CY	NC

Whenever you enter the command RF, the flags are displayed in a row at the beginning of a line in the order shown above. At the end of the list of flags, DEBUG displays a static hyphen (—).

New flag values may be entered as alphabetic pairs, and may be entered in any order. Spaces are not required between the flag entries.

To exit the REGISTER command, press the ENTER key. Flags for which new values were not entered remain unchanged.

If more than one value is entered for a flag, DEBUG returns a DF error message. If a flag code other than those shown above, is entered, DEBUG returns a BF error message. In both cases, flags up to the error in the list are changed; flags at and after the error are not.

Examples: When you enter R, DEBUG displays all registers, flags, and the decoded instruction for the current location.

When you enter RF, DEBUG displays the flags:

```
NV UP DI NG NZ AC PE NC — _
```

Enter any valid flag designations, in any order, with or without spaces. For example, enter:

```
NV UP DI NG NZ AC PE NC — PLEICY
```

DEBUG responds with only the DEBUG prompt. To see the changes, enter the DUMP, REGISTER, or RF command:

```
RF
NV UP EI PL NZ AC PE CY — _
```

Press ENTER to leave the flags this way, or to enter different flag values.

SEARCH (S) Command

Purpose: To search the specified *range* for the specified list of bytes.

Syntax: S *range list*

Comments: The list may contain one or more bytes, each separated by a space or comma.

If the list contains more than one byte, only the first address of the byte string is returned.

If the list contains only one byte, all addresses of the byte in the *range* are displayed.

Example: Enter:

```
SCS:100 110 41
```

DEBUG displays:

```
04BA:0104
04BA:010D
—_
```

TRACE (T) Command

Purpose: To execute one instruction, and display the contents of all registers, flags, and the decoded instruction.

Syntax: T [=*address*][*value*]

Comments: If the optional =*address* is entered, tracing occurs at the =*address* specified. If =*address* includes the segment designation, then both CS and the instruction pointer are specified. If =*address* omits the segment designation, only the instruction pointer is specified. The optional *value* causes DEBUG to execute and trace the number of steps specified by *value*.

Example: When you enter T, DEBUG returns a display of the registers, flags, and decoded instruction for that one instruction.

Enter:

```
T=011A 10
```

DEBUG executes 16 (10 hex) instructions, beginning at 011A in the current segment, and then displays all registers and flags for each instruction as it is executed. The display scrolls away until the last instruction is executed, then stops, and displays the register and flag *values* for the last few instructions performed. ^NUM LOCK suspends the display at any point, so that the registers and flags can be studied for any instruction.

UNASSEMBLE (U) Command

Purpose: To unassemble bytes and display the source statements that correspond to them, along with *addresses* and byte *values*.

Syntax: U [*address* [*L value*]

U [*range*]

Comments: The display of unassembled code looks like a listing for an assembled file.

If the UNASSEMBLE command is entered without parameters, 20 hexadecimal bytes are unassembled to show corresponding instructions.

If the UNASSEMBLE command is entered with the *range* parameter, DEBUG unassembles all bytes in the *range*, up to 20 hexadecimal bytes.

If there are fewer bytes in the *range* than the number displayed when U is entered without parameters, only the bytes in the *range* are displayed.

If the UNASSEMBLE command is entered with the *address* parameter, DEBUG unassembles the default number of bytes, beginning at the *address* specified.

If the UNASSEMBLE command is entered with the *address* [*L value*] parameters, DEBUG unassembles all bytes, beginning at the *address* specified, for the number of lines specified by *value*.

Entering U with the *address* [*L value*] parameters overrides the default limit (20H bytes).

Example: Enter:

U04BA:100

DEBUG unassembles 16 bytes, beginning at address 04BA:0100:

```
04BA:0100   206472      AND     [SI+72],AH
04BA:0103   69          DB      69
04BA:0104   7665        JBE     016B
04BA:0106   207370      AND     [BP+DI+70],DH
04BA:0109   65          DB      65
04BA:010A   63          DB      63
04BA:010B   69          DB      69
04BA:010C   66          DB      66
04BA:010D   69          DB      69
04BA:010E   63          DB      63
04BA:010F   61          DB      61
```

Enter:

U04BA:0100 0108

The display shows:

```
04BA:0100   206472      AND     [SI+72],AH
04BA:0103   69          DB      69
04BA:0104   7665        JBE     016B
04BA:0106   207370      AND     [BP+DI+70],DH
```

Enter:

U04BA:100 120

The display appears exactly as above with the UCS:100 command.

Enter:

UCS:100 L 20

All the bytes from address CS:100 through address CS:120 are unassembled and displayed. This applies to both display formats.

If the bytes in some addresses are altered, the instruction statements are altered. The UNASSEMBLE command can be entered for the changed locations, the new instructions viewed, and the unassembled code used to edit the source file.

WRITE (W) Command

Purpose: To write the file being debugged to a disk file.

Syntax: W [*address* [*d*: *sector sector*]]

Comments: If W is entered without parameters, BX:CX must already be set to the number of bytes to be written; the file is written beginning from CS:100.

If the WRITE command is given with only an *address*, the file is written beginning at that *address*.

If a GO or TERMINATE command is given, BX:CX must be reset before using the WRITE command without parameters.

If a file is loaded and modified, the name, length, and starting address are all set correctly to save the modified file, as long as the length has not changed.

The file must have been named, either during DEBUG invocation, or with the NAME command. Both the invocation and the NAME command format a file-name properly in the normal format of a file control block at CS:5C.

If the WRITE command is given without parameters, BX:CX must be set to the number of bytes to be written. DEBUG writes the BX:CX number of bytes to the disk file. The debugged file is written to the diskette from which it was loaded, and is written over the original file.

If the WRITE command is given with parameters, the write begins from the memory *address* specified; the file is written to the drive specified. The drive specifier is numeric: 0=A: or 1=B:. DEBUG writes the file beginning at the logical *sector* number specified by the first *sector*; and continues until the number of sectors specified in the second *sector* have been written.

NOTE: Writing to absolute sectors is extremely dangerous because the process bypasses the file handler.

Example: Enter:

```
W
```

DEBUG writes out the file to diskette, and displays the DEBUG prompt:

```
W
—_
```

Enter:

```
WCS:100 1 37 2B
```

DEBUG writes out the contents of memory, beginning with the address CS:100, to the diskette in Drive B:. The data written out starts in disk logical sector number 37H, and consists of 2BH sectors. When the write is complete, DEBUG displays the prompt:

```
WCS:100 1 37 2B
—
```

5-4. DEBUG ERROR MESSAGES

During the DEBUG session, any of the following error messages may appear. Each error terminates the DEBUG command with which it is associated, but does not terminate DEBUG itself.

Error Code:	**Definition:**
BF	Bad flag. Retry the REGISTER (RF) command with the correct flag code setting.
BP	Breakpoints. The GO command can have no more than 10 breakpoints.
BR	Bad register. Retry the REGISTER (R) command with a correct register name.
DF	Double flag. A single flag had conflicting codes specified. A flag is permitted to be changed only once per REGISTER (RF) command.

6-1. OVERVIEW

Capabilities of the Linker Utility

LINK is a virtual linker, which can link programs too large for available memory.

LINK produces relocatable executable object code.

LINK handles user-defined overlays.

LINK prompts the operator for input and output modules and other link session parameters.

LINK runs with an automatic response file to answer the linker prompts.

The output file from LINK (run file) is not bound to specific memory addresses and, therefore, can be loaded and executed at any convenient address by the operating system.

LINK uses a dictionary-indexed library search method, substantially reducing link time for sessions involving library searches.

LINK is capable of linking files totaling 384 Kbytes.

LINK Operation Summary

LINK combines several object modules into one relocatable load module, or run file.

As it combines modules, LINK resolves external references between object modules and can search multiple library files for definitions for any external references left unresolved.

LINK also produces a list file that shows external references resolved and any error messages.

LINK uses available memory as much as possible. When available memory is exhausted. LINK then creates a temporary disk file (VM.TMP) and becomes a virtual linker.

A block diagram of LINK operation is shown in Figure 6-1.

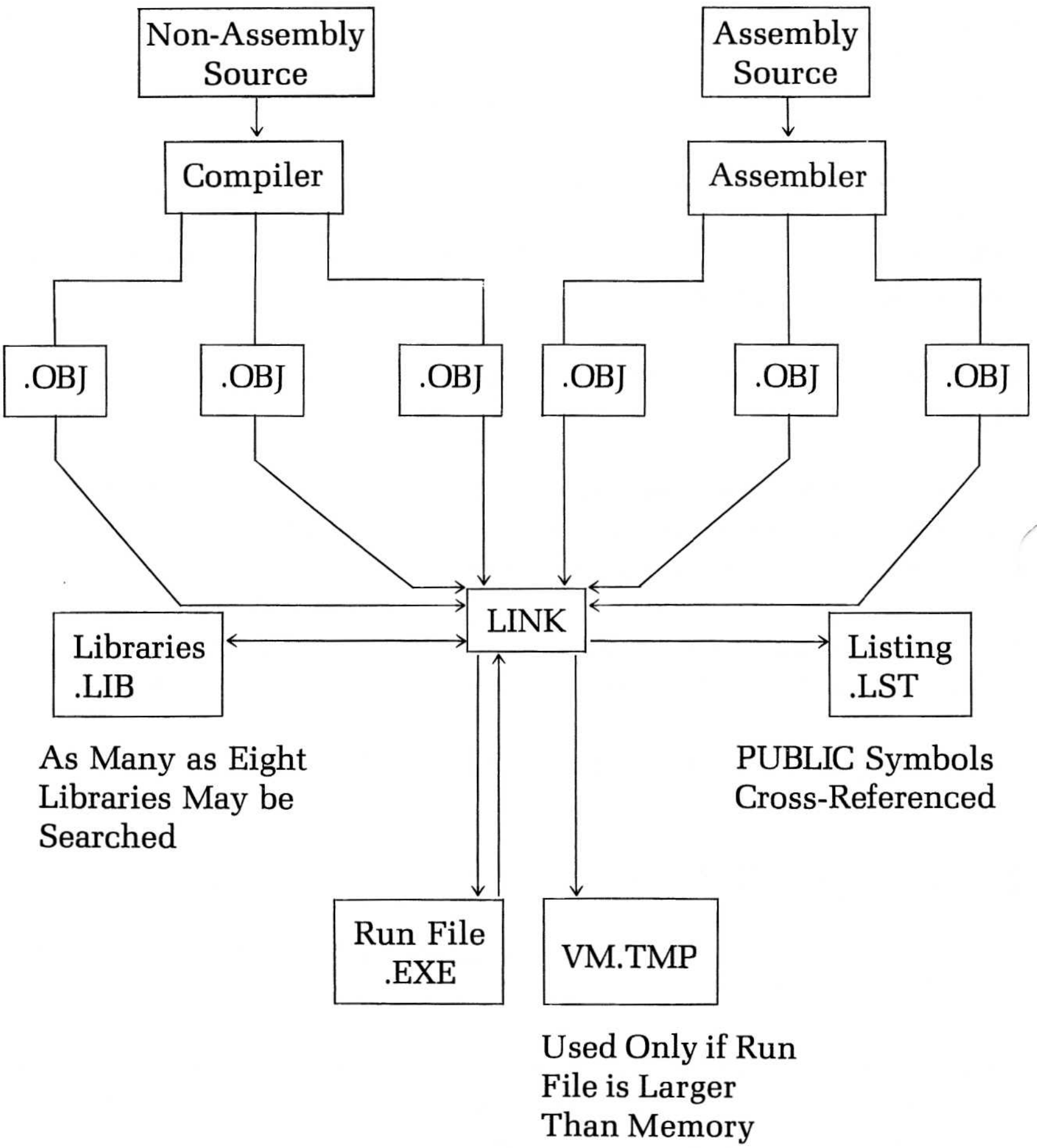


Figure 6-1. LINK Operation Block Diagram.

Definitions

Three important terms appear in this chapter. These terms describe the underlying functioning of LINK. An understanding of the concepts that define these terms provides a basic understanding of the way LINK works.

1. Segment. A segment is a contiguous area of memory up to 64 Kbytes in length. A segment may be located anywhere in memory on a paragraph (16-byte) boundary. The contents of a segment are addressed by a segment-register/offset pair.

2. Group. A group is a collection of segments which fit within 64 Kbytes of memory. The segments are named according to group by the assembler, or compiler.

   The group is used for addressing segments in memory. Each group is addressed by a single segment register. The segments within the group are addressed by the segment register plus an offset. LINK verifies that the object modules of a group meet the 64-Kbyte constraint.

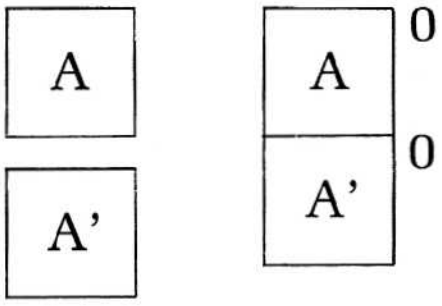
3. Class. A class is a collection of segments. The naming of segments to a class controls the order and relative placement of segments in memory. The class name is specified by the assembler or compiler. Segments of a class are loaded into memory contiguously. The segments are ordered within a class in the order LINK encounters the segments in the object files. One class precedes another in memory only if a segment for the first class precedes all segments for the second class in the input to LINK. Classes may be loaded across 64-Kbyte boundaries. The classes are divided into groups for addressing.

6-2. COMBINING AND ARRANGING SEGMENTS

LINK works with four combine types, which are declared in the source module for the assembler or compiler: private, public, stack, and common.

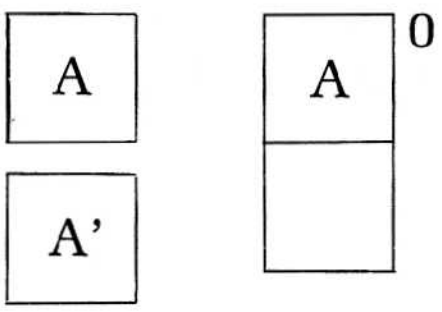
LINK combines segments for these combine types as follows:

Private



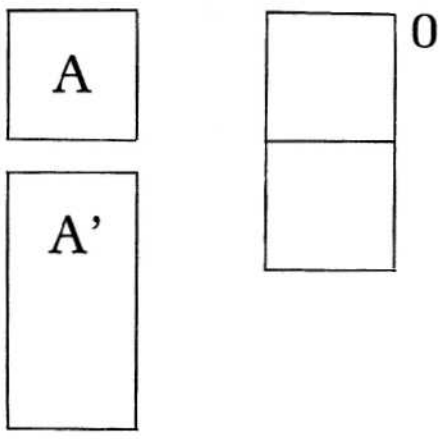
Private segments are loaded separately and remain separate. They may be physically contiguous, but not logically, even if the segments have the same name. Each private segment has its own base address.

Public



Public segments of the same name and class name are loaded contiguously. Offset is from beginning of first segment loaded through last segment loaded.There is only one base address for all public segments of the same name and class name. Combine types stack and memory are treated the same as public; however, the stack pointer (SP) is set to the first address of the first stack segment.

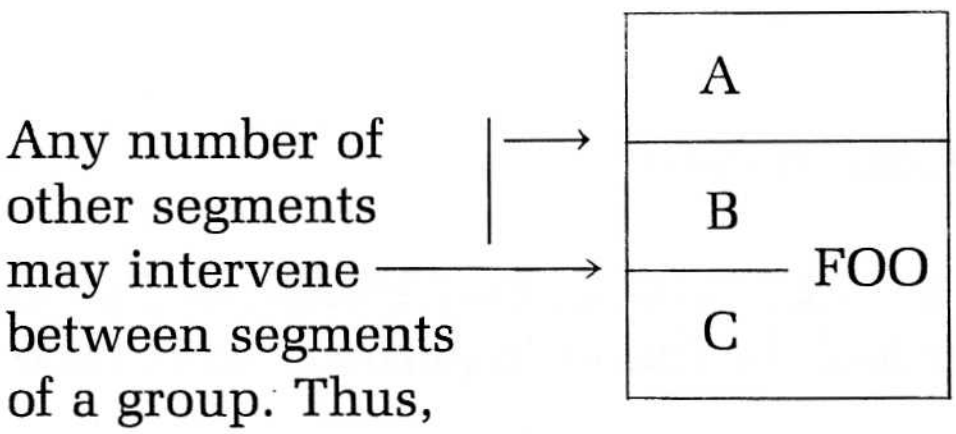
Common



Common segments of the same name and class name are loaded, overlapping one another. There is only one base address for all common segments of the same name. The length of the common area is the length of the longest segment.

Placing segments in a group in the assembler provides offset addressing of items from a single base address for all segments in that group.

DS:DGROUP ⟶ XXXX0H 0 — Relative offset



Any number of other segments may intervene between segments of a group. Thus, the offset of FOO may be greater than the size of segments in group combined, but no larger than 64 Kbytes.

An operand of DGROUP: FOO returns the offset of FOO from the beginning of the first segment of DGROUP (here, segment A).

Segments are grouped by declared class names. LINK loads all the segments belonging to the first class name encountered, then loads all the segments of the next class name encountered, and so on until all classes have been loaded.

If your program contains:	They will be loaded as:
A SEGMENT 'FOO'	'FOO'
B SEGMENT 'BAZ'	A
C SEGMENT 'BAZ'	E
D SEGMENT 'ZOO'	'BAZ'
E SEGMENT 'FOO'	B
	C
	'ZOO'
	D

When writing assembly language programs, control over the ordering of classes in memory is done by writing a dummy module. After writing the module list it first after the LINK Object Modules prompt. The dummy module declares segments into classes in the order you want the classes loaded.

CAUTION

Do not use this method with BASIC, COBOL, FORTRAN, or Pascal programs. Allow the compiler and the linker to perform their tasks in the normal way.

For example:

```
A    SEGMENT   'CODE'
A    ENDS
B    SEGMENT   'CONST'
B    ENDS
C    SEGMENT   'DATA'
C    ENDS
D    SEGMENT   STACK        'STACK'
D    ENDS
E    SEGMENT   'MEMORY'
E    ENDS
```

Caution should be taken to declare all classes to be used in your program in this module. If you do not, you lose absolute control over the ordering of classes. To have memory combine type to be loaded as the last segments of your program, you can use this method.

Simply add MEMORY between SEGMENT and 'MEMORY' in the E segment line above. Note, however, that these segments are loaded last only because you imposed this control on them, not because of any inherent capability in the linker or assembler operations.

6-3. FILES USED BY LINK

LINK works with one or more input files, producing two output files. Additionally, LINK may create a virtual memory file, and may be directed to search one to eight library files. For each type of file, you may give a three-part file specification. The format for LINK file specifications is:

d:filename.ext

where:

d: is the drive specifier. The colon is always required as part of the drive specifier.

filename is any legal filename of one to eight characters.

.ext is an one-to-three character extension to the filename. The period is always required as part of the extension.

LINK Input Files

If no extensions are given in the input (object) file specifications, LINK recognizes by default:

File:	Default Extension:
Object	.OBJ
Library	.LIB

LINK Output Files

LINK appends to the output (run and list) files the following default extensions:

File:	Default Extension:
Run	.EXE (may not be overridden)
List	.MAP (may be overridden)

VM.TMP File

LINK uses available memory for the link session. If the files to be linked create an output file that exceeds available memory, LINK creates a temporary file and names it VM.TMP. If LINK needs to create VM.TMP, it displays the message:

```
VM.TMP has been created
Do not change diskette in drive (A: or B:)
```

Once this message is displayed, you must not remove the diskette from the default drive until the link session ends. If the diskette is removed, the operation of LINK is unpredictable, and LINK might return the error message:

```
Unexpected end of file on VM.TMP
```

LINK uses VM.TMP as a virtual memory. The contents of VM.TMP are subsequently written to the file named following the Run File: prompt. VM.TMP is a working file only and is deleted at the end of the linking session.

CAUTION

Do not use VM.TMP as a file name for any file. If the user has a file named VM.TMP on the default drive and LINK requires the VM.TMP file, LINK will delete the VM.TMP on disk and create a new VM.TMP. Thus, the contents of the previous VM.TMP file will be lost.

6-4. RUNNING LINK

Running LINK requires two types of commands: a command to invoke LINK and a command to answer command prompts. Six switches control alternate LINK features.

Command Prompts

LINK is commanded by entering responses to four text prompts. When you have entered a response to the current prompt, the next appears. When the last prompt has been answered, LINK begins linking automatically without further command. When the link session is finished, LINK exits to the operating system. If the link session is unsuccessful, LINK returns the appropriate error message.

LINK prompts you for the names of object, run, list files, and for libraries. The prompts are listed in their order of appearance. For prompts which can default to preset responses, the default response is shown in square brackets ([ ]) following the prompt. The Object Modules: prompt is followed by only a filename extension default response because it has no preset filename response and requires a filename from the operator.

Object Modules [.OBJ]:

Enter a list of the object modules to be linked. LINK assumes, by default, that the filename extension is .OBJ. If an object module has any other filename extension, the extension must be given here. Otherwise, the extension may be omitted.

Modules must be separated by blank spaces or plus signs (+).

Remember that LINK loads segments into classes in the order encountered. Use this information for setting the order in which the object modules are entered.

Run File [First-Object-filename.EXE]:

The filename entered will be created to store the run (executable) file that results from the link session. All run files receive the filename extension .EXE, even if you specify an extension (the user-specified extension is ignored).

If no response is entered to the Run File: prompt, LINK uses the first filename entered in response to the Object Modules: prompt as the run filename.

Example:

```
Run File [FUN.EXE]: B:PAYROLL/P
```

This response directs LINK to create the run file PAYROLL.EXE on the diskette in Drive B:. Also, LINK will pause, which allows you to insert a new diskette to receive the run file.

List File [NUL.MAP]:

The list file contains an entry for each segment in the input (object) modules. Each entry also shows the offset (addressing) in the run file.

The default response is the NUL filename which specifies that no list file will be created.

Libraries [.LIB]:

The valid responses are zero to eight library filenames. Library files must have been created by a library utility. LINK assumes, by default, that the filename extension is .LIB for library files.

Library filenames must be separated by blank spaces or plus signs (+).

LINK searches the library files in the order listed to resolve external references. When it finds the module that defines the external symbol, LINK processes the module as another object module.

If LINK cannot find a library file on the diskettes in the disk drives, it returns the message:

```
Cannot find library library-name
Enter new drive letter:
```

Simply press the letter for the drive specifier (for example, B).

LINK does not search within each library file sequentially. LINK uses a method called dictionary-indexed library search. This means that it finds definitions for external references by index access rather than searching from the beginning of the file to the end for each reference. This indexed search substantially reduces the link time for any sessions involving library searches.

Switches

The six switches control alternate linker functions. Switches must be entered at the end of a prompt response, regardless of which method is used to invoke LINK. Switches may be grouped at the end of any one of the responses, or may be scattered at the end of several. If more than one switch is entered at the end of one response, each switch must be preceded by a slash (/).

All switches may be abbreviated, from a single letter through the whole switch name. The only restriction is that an abbreviation must be a sequential substring from the first letter through the last entered; no gaps or transpositions are allowed. For example, for the switch /DSALLOCATE:

Legal:	Illegal:
/D	/DSL
/DS	/DAL
/DSA	/DLC
/DSALLOCA	/DSALLOCT

The switches used in LINK are:

/DSALLOCATE

Use of the /DSALLOCATE switch directs LINK to load all data (DGROUP) at the high end of the data segment (DS). Otherwise, LINK loads all data at the low end of DS. At runtime, the DS pointer is set to the lowest possible address and allows the entire DS segment to be used. Use of the /DSALLOCATE switch in combination with the default load low (that is, the /HIGH switch is not used), permits the user application to dynamically allocate any available memory below the area specifically allocated within DGROUP, yet to remain addressable by the same DS pointer. This dynamic allocation is needed for Pascal and FORTRAN programs.

NOTE: Your applications program may dynamically allocate up to 64 Kbytes (or the actual amount available) less the amount allocated within DGROUP.

/HIGH

Use of the /HIGH switch causes LINK to place the run image as high as possible in memory. Otherwise, LINK places the run file as low as possible.

CAUTION

Do not use the /HIGH switch with Pascal or FORTRAN programs.

/LINENUMBERS

Use of the /LINENUMBERS switch directs LINK to include in the list file the line numbers and addresses of the source statements in the input modules. Otherwise, line numbers are not included in the list file.

NOTE: Not all compilers produce object modules that contain line number information. In these cases, of course, LINK cannot include line numbers.

/MAP

/MAP directs LINK to list all public (global) symbols defined in the input modules. If /MAP is not given, LINK will list only errors (which includes undefined globals).

The symbols are listed alphabetically. For each symbol, LINK lists its value and its segment:offset location in the run file. The symbols are listed at the end of the list file.

/PAUSE

The /PAUSE switch causes LINK to pause in the link session when the switch is encountered. Normally, LINK performs the linking session without stopping from beginning to end. This allows you to exchange the diskettes before LINK outputs the run (.EXE) file.

When LINK encounters the /PAUSE switch, it displays the message:

```
About to generate .EXE file
Change disks hit any key
```

LINK resumes processing when you press any key.

CAUTION

Do not exchange the diskette which will receive the list file, or the diskette used for theVM.TMP file, if created.

/STACK:*number*

number represents any positive numeric value (in hexadecimal radix) up to 65536 bytes. If the /STACK switch is not used for a link session, LINK calculates the necessary stack size automatically.

If a value from 1 to 511 is entered, LINK uses 512.

All compilers and assemblers should provide information in the object modules that allow the linker to compute the required stack size.

At least one object (input) module must contain a stack allocation statement. If not, LINK will return a WARNING: NO STACK STATEMENT error message.

Invoking LINK

LINK may be invoked in three ways. By the first method, you enter the commands in response to individual prompts. By the second method, you enter all commands on the line used to invoke LINK. By the third method, you create a response file that contains all the necessary commands.

Summary of Methods Used to Invoke LINK:

Method 1	LINK
Method 2	LINK *filenames[/switches]*
Method 3	LINK @ *filename*

Method 1, LINK:

Enter:

```
LINK
```

LINK will be loaded into memory. Then, LINK returns a series of four text prompts that appear one at a time. Answer the prompts as commands to LINK to perform specific tasks.

When entering the LINK command, you may enter one or more switches, each of which must be preceded by a slash mark. If a switch is not included, LINK defaults to not performing the function described for the switches Tables 6-1 and 6-2.

Table 6-1.
Command Prompt Summary

Prompt:	Responses:
Object Modules [.OBJ]:	List .OBJ files to be linked, separated by a blank space or plus sign. If a plus sign is the last character entered, the prompt will reappear. No default response is required
Run File [object-file.EXE]:	*filename* for executable object code. The default is first-object-filename. EXE.
List File [NUL.MAP]:	*filename* for listing. The default is RUN *filename*.
Libraries [.LIB]:	List filenames to be searched, separated by blank spaces or plus signs. If the plus sign is last character entered, the prompt will reappear. The default is no search.

Table 6-2.
Linker Switch Parameters

Switch:	Action:
/DSALLOCATE	Load data at high end of data segment. Required for Pascal and FORTRAN programs.
/HIGH	Place run file as high as possible in memory. Do not use with Pascal or FORTRAN programs.
/LINENUMBERS	Include line numbers in list file.

Continued

Continued

/MAP	List all global symbols with definitions.
/PAUSE	Halt linker session and wait for the ENTER key to be pressed.
/STACK:number	Set fixed stack size in run file.

Command Characters

LINK provides three command characters.

+ Use the plus sign (+) to separate entries and to extend the current physical line following the Object Modules and Libraries prompts. A blank or space may be used to separate object modules.

To enter a large number of responses (each which may also be very long), enter a plus sign at the end of the physical line (to extend the logical line). If the plus sign is the last entry following these two prompts, LINK will prompt you for more modules names.

When the Object Modules or Libraries prompt appears again, continue to enter responses. When all the modules to be linked have been listed, verify that the response line ends with a module name; not a plus sign.

Examples:

```
Object Modules [.OBJ]: FUN TEXT TABLE CARE
Object Modules [.OBJ]: FOOD+JUNQUE
Object Modules [.OBJ]: CORSAIR
```

; Enter a single semicoln (;) at any time after the first prompt (from Run File on) to select default

responses to the remaining prompts. This feature saves time.

NOTE: Once the semicolon has been entered, you can no longer respond to any of the prompts for that link session. Therefore, do not use the semicolon to skip over some prompts.

Example:

```
Object Modules [.OBJ]: FUN TEXT TABLE CARE
Run Module [FUN.EXE]:;              (ENTER)
```

The remaining prompts will not appear, and LINK will use the default values (including FUN.MAP for the List File).

^BREAK — Use ^BREAK at any time to abort the link session. If you enter an erroneous response, such as the wrong filename or an incorrectly spelled filename, you must press ^BREAK to exit LINK, then reinvoke LINK and start over. If the error has been typed, but not entered, you may delete the erroneous characters on that line only.

Method 2, LINK *filenames*[/*switches*:]:

Enter: LINK *object-list*,*runfile*,*listfile*,*lib-list*[/*switch* . . .]

The entries following LINK are responses to the command prompts. The entry fields for the different prompts must be separated by commas.

where: *object-list* is a list of object modules, separated by plus signs.

runfile is the name of the file to receive the executable output.

listfile is the name of the file to receive the listing.

lib-list is a list of library modules to be searched.

/switch are optional switches, which may be placed following any of the response entries (just before any of the commas or after the *lib-list*, as shown).

To select the default for a field, simply enter a second comma without spaces in between (see the example below).

Example:

```
LINK FUN+TEXT+TABLE+CARE/P/M,,FUNLIST, COBLIB.LIB
```

This example causes LINK to be loaded, then causes the object modules FUN.OBJ, TEXT.OBJ, TABLE.OBJ, and CARE.OBJ to be loaded. LINK then pauses (caused by the /P switch). When you press any key, LINK links the object modules, produces a global symbol map (the /M switch), defaults to FUN.EXE run file, creates a list file named FUNLIST.MAP, and searches the library file COBLIB.LIB.

Method 3, LINK *filespec*:

Enter: LINK *filespec*

where: *filespec* is the name of a response file. A response file contains answers to the LINK prompts (shown under method 1 for invoking), and may also contain any of the switches. Method 3 permits you to conduct the LINK session without interactive (direct) responses to the LINK prompts.

NOTE: Before using method 3 to invoke LINK, you must first create the response file.

A response file has text lines, one for each prompt. Responses must appear in the same order as the command prompts appear.

Use switches and command characters in the response file in the same way as they are used for responses entered on the keyboard.

When the LINK session begins, each prompt is displayed in turn with the responses from the response file. The response file must contain valid responses for all the prompts, either in the form of filenames, the semicolon command character, or when LINK prompts for a response not found in the response file, it waits for a keyboard entry. When a legal response has been entered, LINK continues the link session.

Example:

```
FUN TEXT TABLE P/M
FUN
FUNLIST
COBLIB.LIB
```

This response file causes LINK to load the four object modules. LINK pauses before creating and producing a public symbol map to permit you to exchange diskettes (see discussion under /PAUSE).

When you press any key, the output files will be named FUN.EXE and FUNLIST.MAP. LINK will search the library file COBLIB.LIB, and will use the default settings for the flags.

6-5. ERROR MESSAGES

All errors cause the link session to abort. Therefore, after the cause is found and corrected, LINK must be rerun.

Error Message: ATTEMPT TO ACCESS DATA OUTSIDE OF SEGMENT BOUNDS, POSSIBLY BAD OBJECT MODULE

Cause: Probably a bad object file.

Error Message: BAD NUMERIC PARAMETER

Cause: Numeric value not in digits.

Error Message: CANNOT OPEN TEMPORARY FILE

Cause: LINK is unable to create the file VM.TMP because the diskette directory is full.

Cure: Insert a new diskette. Do not change the diskette that will receive the list.MAP file.

Error Message: ERROR: DUP RECORD TOO COMPLEX

Cause: DUP record in assembly language module is too complex.

Cure: Simplify DUP record in assembly language program.

Error Message: ERROR: FIXUP OFFSET EXCEEDS FIELD WIDTH

Cause: An assembly language instruction refers to an address with a short instruction instead of a long instruction.

Cure: Edit assembly language source and reassemble.

Error Message: INPUT FILE READ ERROR

Cause: Probably a bad object file.

Error Message: INVALID OBJECT MODULE

Cause: Object module(s) incorrectly formed or incomplete (as when assembly was stopped in the middle).

Error Message: SYMBOL DEFINED MORE THAN ONCE

Cause: LINK found two or more modules that define a single symbol name.

Error Message: PROGRAM SIZE OR NUMBER OF SEGMENTS EXCEEDS CAPACITY OF LINKER

Cause: The total size may not exceed 384 Kbytes and the number of segments may not exceed 255.

Error Message: REQUESTED STACK SIZE EXCEEDS 64K

Cure: Specify a size < or = 64 Kbytes with the /STACK switch.

Error Message: SEGMENT SIZE EXCEEDS 64K

Comment: 64 Kbytes is the addressing system limit.

Error Message: SYMBOL TABLE CAPACITY EXCEEDED

Cause: Too many long names entered, exceeding approximately 25 Kbytes.

Error Message: TOO MANY EXTERNAL SYMBOLS IN ONE MODULE

Comment: The limit is 256 external symbols per module.

Error Message: TOO MANY GROUPS

Comment: The limit is 10 groups.

Error Message: TOO MANY LIBRARIES SPECIFIED

Comment: The limit is eight.

Error Message: TOO MANY PUBLIC SYMBOLS

Comment: The limit is 1024.

Error Message: TOO MANY SEGMENTS OR CLASSES

Comment: The limit is 256 (segments and classes takentogether).

Error Message: UNRESOLVED EXTERNALS: list

Cause: The external symbols listed have no defining module among the modules or libraries files specified.

Error Message: VM READ ERROR

Cause: A diskette problem; not LINK-caused.

Error Message: WARNING: NO STACK SEGMENT

Cause: None of the object modules specified contains a statement allocating stack space, but you entered the /STACK switch.

Error Message: WARNING: SEGMENT OF ABSOLUTE OR UNKNOWN TYPE

Cause: A bad object module or an attempt to link modules LINK cannot handle (for example, an absolute object module).

Error Message: WRITE ERROR IN TMP FILE

Cause: No more diskette space remaining to expand VM.TMP file.

Error Message: WRITE ERROR ON RUN FILE

Cause: Usually, not enough diskette space for run file.

DOS FILE CONTROL BLOCK (FCB) DEFINITION

The DOS file control block (FCB) is defined as follows:

Byte 0 Drive code. The drive codes are specified by 0= default drive, 1=drive A:, and 2=drive B:.

Bytes 1-8 Filename. If the filename consists of fewer than eight characters, the name must be left-justified with trailing blanks.

Bytes 9-11 Extension to filename. If the extension is fewer than three characters, it must be left-justified with trailing blanks. The extension can also consist of all blanks.

Bytes 12-13 Current block (extent). This word (low byte first) specifies the current block of 128 records, at the start of the file, in which sequential disk reads and writes occur. If zero, then the first block of the file is being accessed; if one, then the second block of the file is being accessed, and so on. Combined with the current record field (byte 32), a particular logical record is identified.

Bytes 14-15 Size of the record with which the user wishes to work. This word may be filled immediately after opening the file if the default logical record size (128 bytes) is not desired. The OPEN and CREATE functions set this field to 128; it is also changed to 128 if a READ or WRITE is attempted with the field set to zero.

Bytes 16-19 File size. This is the current size, in bytes, of the file. It may be read by user programs, but must not be written by them.

Bytes 20-21 Date. This is normally the date of the last WRITE to the file. It is set by all disk WRITE and CREATE operations to today's date. It is set by OPEN to the date recorded in the disk directory for the file. User programs may modify this field after writing to a file (but before closing it) to changes the date recorded in the disk directory.

The format of this 16-bit field is as follows: bits 0-4, day of the month; bits 5-8, month of the year; bits 9-15, the year minus 1980. All zeros mean no date.

Bytes 22-23 Time. Similar to date, above. The format is bits 0-4, seconds; bits 5-10, minutes; bits 11-15, hours.

Bytes 24-31 Reserved for DOS.

Byte 32 Current record. This identifies the record within the current block of 128 records that are accessed with a sequential READ or WRITE function. See current block, bytes 12-13.

Bytes 33-36 Random record. This field must only be set if the file is to be accessed with a random READ or WRITE function. If the record size is greater than or equal to 64 bytes, only the first three bytes are used as a 24-bit number. This represents the position in the file of a record. If the record size is less than 64 bytes, all four bytes are used as a 32-bit number for the same purpose. This field is large enough to address any byte in a file of the maximum size, 230 bytes.

The Extended FCB

The extended FCB is a special format used to search for files in the disk directory with special attributes. It consists of seven bytes in front of a normal FCB, formatted as follows:

FCB-7 Flag. FF hex is placed here to signal an extended FCB.

FCB-6 - FCB-2 Reserved.

FCB-1 Attribute byte. If bit 1 = 1, hidden files are included in directory searches. If bit 2 = 1, system files are included in directory searches.

Any reference to an FCB in the description of DOS function calls, whether opened or unopened, may use either a normal FCB or an extended FCB. A normal FCB has the same effect as an extended FCB with the attribute byte set to zero.

DOS INTERRUPTS AND FUNCTION CALLS

Interrupts

DOS reserves interrupt types 20 to 3F hex for its use. This means absolute locations 80 to FF hex are the transfer address storage locations reserved by DOS. The defined interrupts (with all values in hex) are as follows:

20 Program terminate. This is the normal way to exit a program. This vector transfers to the logic in the DOS for restoration of ^BREAK exit addresses to the values they had on entry to the program. All file buffers are flushed to disk. All files that have changed in length should have been closed (see function call 10 hex) prior to issuing this interrupt. If the changed file was not closed, its length will not be recorded correctly in the directory. When this interrupt is executed, CS **must** point to the 100H parameter area.

21 Function request. See the Function Requests section.

22 Terminate address. The address represented by this interrupt (88-8B hex) is the the address to which control will transfer when the program terminates. This address is copied into low memory of the segment the program is loaded into at the time this segment is created. If a program wishes to execute a second program, it must set the terminate address prior to creation of the segment that the program will be loaded into. Otherwise, once the second program executes, its termination would cause transfer to its host's termination address.

23 ^BREAK exit address. If you type ^BREAK during keyboard input or video output, "C" will be displayed on the screen and an interrupt type 23 hex will be executed. If the

^BREAK routine preserves all registers, it may end with a return-from-interrupt instruction (IRET) to continue program execution. If functions 9 or 10 (buffered input/output) were being executed, then I/O will continue from the start of the line. When the interrupt occurs, all registers are set to the values that they had when the original call to DOS was made. There are no restrictions on what the ^BREAK handler is allowed to do, including DOS function calls, as long as the registers remain unchanged if IRET is used.

If the program creates a new segment and loads in a second program which changes the ^BREAK address, the termination of the second program and the return to the first will cause the ^BREAK address to be restored to the value it had prior to the execution of the second program.

24 Fatal error abort vector. When a fatal error occurs within DOS, control is transferred with an INT 24H. On entry to the error handler, AH will have its bit 7=0 if the error was a hard disk error (probably the most common occurrence); bit 7=1, if it is not a hard disk error. If it is a hard disk error, bits 0-2 include the following:

Bit 0 is 0 if read; 1, if write.

Bit 2:	**Bit 1:**	**Affected Disk Area:**
0	0	Reserved Area
0	1	File Allocation Table
1	0	Directory
1	1	Data Area

AL, CX, DX, and DS:BX will be setup to perform a retry of the transfer with INT 25H or INT 26H (below). DI will have a 16-bit error code returned by the hardware.

The values returned are:

0	Write Protect
2	Disk Not Ready
4	Data Error
6	Seek Error
8	Sector Not Found
A	Write Fault
C	General Disk Failure

The registers will be set up for a BIOS disk call and the returned code will be in the lower half of the DI register with the upper half undefined. The user stack will look as follows from top to bottom:

IP	Registers such that if an IRET is executed
CS	the DOS will respond according to (AL)
FLAGS	as follows:

(AL)=0	Ignore the Error
(AL)=1	Retry the Operation
(AL)=2	Abort the Program

CAUTION

If the =1 option is used, stack DS, BX, CX, and DX must not be used.

AX	User Registers at the Time of Request
BX	
CX	
DX	

SI

DI

BP

DS

ES

IP The Interrupt From the User to DOS

CS

FLAGS

Currently, the only error possible when AH bit 7=1 is a bad memory image of the file allocation table.

25 Absolute disk read. This transfers control directly to the DOS BIOS. Upon return, the original flags are still on the stack (put there by the INT instruction). This is necessary because return information is passed back in the flags. Be sure to pop the stack to prevent uncontrolled growth. For this entry point, records and sectors are the same size. The request is as follows:

(AL) Drive Number (0=A; 1=B)

(CX) Number of Sectors to Read

(DX) Beginning Logical Record Number

(DS:BX) Transfer Address

The number of records specified are transferred between the given drive and the transfer address. Logical record numbers are obtained by numbering each sector sequentially, starting

from zero and continuing across track boundaries. For example, logical record number 0 is track 0 sector 1; whereas logical record number 12 hex is track 2 sector 3.

All registers except for the segment registers are destroyed by this call. If the transfer is successful, the carry flag (CF) is zero. If the transfer is not successful, CF=1 and (AL) is indicate the error as follows:

Return (Hex):	**Description:**
80	Attachment Failed to Respond.
40	Seek Operation Failed.
20	Controller Failure.
10	Bad CRC on Diskette Read.
08	DMA Over-Run on Operation.
04	Requested Sector Not Found.
03	Write Attempt on Write-Protected Diskette.
02	Address Mark Not Found.

26 Absolute disk write. This vector is the counterpart to interrupt 25 above. Except for the fact that this is a write, the description above applies.

27 Terminate, but stay resident. This vector is used by programs which are to remain resident when COMMAND regains control. Such a program is loaded as an executing COM file by COMMAND. After it has initialized itself, it must set DX to its last address plus one in the segment it is executing in, then execute an interrupt 27H. COMMAND will then treat the program as an extension of DOS, and the program will not be overlaid when other programs are executed.

Function Requests

The user requests a function by placing a function number in the AH register, supplying additional information in other registers as necessary for the specific function, and then executing an interrupt

type 21 hex. When DOS takes control, it switches to an internal stack. User registers, except AX, are preserved unless information is passed back to the requester, as indicated in the specific requests. The user stack needs to be sufficient to accommodate the interrupt system. It is recommended that it be 80 hex, in addition to the user needs. There is an additional mechanism provided for programs that conform to CP/M calling conventions. The function number is placed in the CL register, other registers are set as normal according to the function specification, and an intrasegment call is made to location 5 in the current code segment. This method is only available to functions which do not pass a parameter in AL and whose numbers are equal to or less than 36. Register AX is always destroyed if this mechanism is used; otherwise, it is the same as normal function requests. The functions are as follows with all values in hex:

0 Program terminate. The terminate and ^BREAK exit addresses are restored to the values they had on entry to the terminating program. All file buffers are flushed, but files which have been changed in length, but not closed, will not be recorded properly in the disk directory. Control transfers to the terminate address.

1 Keyboard input. Waits for a character to be typed at the keyboard, then echoes the character to the video display and returns it in AL. The character is checked for a ^BREAK. If this key is detected, an interrupt 23 hex will be executed.

2 Video output. The character in DL is output to the video display. If a ^BREAK is detected after the output an interrupt, 23 hex is executed.

3 Auxiliary input. Waits for a character from the auxiliary input device, then returns that character in AL.

4 Auxiliary output. The character in DL is output to the auxiliary device.

5 Printer output. The character in DL is output to the printer.

6 Direct console I/O. If DL is FF hex, the AL returns the keyboard input character, if one is ready; otherwise, it returns 00. If DL is not FF hex, then DL is assumed to have a valid character which is output to the video display.

7 Direct console input. Waits for a character to be typed at the keyboard, then returns the character in AL. As with function 6, no checks are made on the character.

8 Console input without echo. This function is identical to function 1, except that the key is not echoed.

9 Print string. On entry, DS:DX must point to a character string in memory terminated by a $ (24 hex). Each character in the string is output to the video display in the same form as function 2.

A Buffered keyboard input. On entry, DS:DX points to an input buffer. The first byte specifies the number of characters the buffer can hold. The first byte cannot be zero. Characters are read from the keyboard and placed in the buffer beginning at the third byte. Reading the keyboard and filling the buffer continues until ENTER is pressed. If the buffer fills to one less than the maximum, then additional keyboard input is ignored until ENTER is pressed. The second byte of the buffer is set to the number of characters received, excluding the carriage return (0D hex) which is always the last character.

B Check keyboard status. If a character is available from the keyboard, AL will be FF hex; otherwise, AL will be 00.

C Character input with buffer flush. First the keyboard type-ahead buffer is emptied. Then if AL is 1, 6, 7, 8, or 0A hex, the corresponding DOS input function is executed. If AL is not one of these values, no further operation is done, and AL returns 00.

D Disk reset. Flushes all file buffers. Unclosed files that have been changed in size will not be properly recorded in the disk directory until they are closed. It is not necessary to call this function before a disk change if all files already written have been closed.

E Select disk. The drive specified in DL (0=A, 1=B, and so on) is selected as the default disk. The number of drives is returned in AL.

F Open file. On entry, DS:DX points to an unopened file control block (FCB). The disk directory is searched for the named file and AL returns FF hex if it is not found. If it is found, AL will return a 00 and the FCB is filled as follows:

If the drive code was 0 (default disk), it is changed to actual disk used (A=1, B=2, and so on). This allows changing the default disk without interfering with subsequent operations on this file. The high byte of the current block field is set to zero. The size of the record to be worked with (FCB bytes E-F hex) is set to the system default of 80 hex. The size of the file and the time and date are set in the FCB from information obtained from the directory.

It is your responsibility to set the record size (FCB bytes E-F) to the size you wish to think of the file in terms of, if the default 80 hex is not appropriate. It is also the your responsibility to set the random record field and/or current block and record fields.

10 Close file. This function must be called after file writes to ensure that all directory information is updated. On entry, DS:DX points to an opened FCB. The disk directory is searched, and if the file is found, its position is compared with that kept in the FCB. If the file is not found in the directory, it is assumed the disk has been changed and AL returns FF hex. Otherwise, the directory is updated to reflect the status in the FCB and AL returns 00.

11 Search for the first entry. On entry, DS:DX points to an unopened FCB. The disk directory is searched for the first matching name (the name can have ?s, indicating any letter matched) and if none are found, AL returns FF hex. Otherwise, locations at the disk transfer address are set as follows:

1. If the FCB provided for searching is an extended FCB, then the first byte is set to FF hex, then 5 bytes of zeros, then the attribute byte from the search FCB, then the drive number used (A=1, B=2, and so on), and then the 32 bytes of the directory entry. Thus the disk transfer address contains a valid unopened extended FCB with the same search attributes as the search FCB.

2. If the FCB provided for searching was a normal FCB, then the first byte is set to the drive number used (A=1, B=2, and so on), and the next 32 bytes contain the matching directory entry. Thus, the disk transfer address contains a valid unopened normal FCB.

Directory entries are formatted as follows:

Location:	**Bytes:**	**Description:**
0	11	Filename and extension.
11	1	Attributes. Bits 1 or 2 make the file hidden.
12	10	Reserved.
22	2	Time. Bits 0-4 = secs*2 5-10 = min 11-15 = hrs
24	2	Date. Bits 0-4 = day 5-8 = month 9-15 = year
26	2	First allocation unit.
28	4	File size, in bytes (30 bits maximum).

12 Search for the next entry. After function 11 has been called and has found a match, function 12 may be called to find the next match to an ambiguous request (?s in the search filename). Both inputs and outputs are are the same as function 11. The reserved area of the FCB keeps information necessary for continuing the search, so it must not be modified.

13 Delete file. On entry, DS:DX points to an unopened FCB. All matching directory entries are deleted. If no directory entries match, AL returns FF; otherwise, AL returns 00.

14 Sequential read. On entry, DS:DX points to an opened FCB. The record addressed by the current block (FCB bytes C-D) and the current record (FCB byte 1F) is loaded at the disk transfer address, then the record address is incremented. If end-of-file is encountered, AL returns either 01 or 03. A return of 01 indicates no data in the record, 03 indicates a partial record is read and filled out with zeros. A return of 02 means there is not enough room in the disk transfer segment to read one record, so the transfer is aborted. AL returns 00 if the transfer is completed successfully.

15 Sequential write. On entry, DS:DX points to an opened FCB. The record addressed by the current block and current record fields is written from the disk transfer address (or, in the case of records less than sector sizes, it is buffered-up for an eventual write when a sector-worth of data is accumulated). The record address is then incremented. If the disk is full, AL returns with a 01. A return of 02 means there is not enough room in the disk transfer segment to write one record, so the transfer is aborted. AL returns 00 if the transfer is completed successfully.

16 Create file. On entry, DS:DX points to an unopened FCB. The disk directory is searched for an empty entry, and AL returns FF if none is found. Otherwise, the entry is initialized to a zero-length file, the file is opened (see function F), and AL returns 00.

17 Rename file. On entry, DS:DX points to a modified FCB which has a drive code and file name in the usual position, and a second file name starting six bytes after the first (DS:DX+11 hex) in what is normally a reserved area. Every matching occurrence of the first is changed to the second (with the restriction that two files cannot have the exact same name and extension). If ?s appear in the second name, then the corresponding positions in the original name will be unchanged. AL returns FF hex if no match is found; otherwise, AL returns 00.

18 Not used.

19 Current disk. AL returns with the code of the current default drive (0=A, 1=B, and so on).

1A Set disk transfer address. The disk transfer address is set to DS:DX. DOS will not allow disk transfers to wrap around within the segment, nor to overflow into the next segment.

1B Allocation table address. On return, DS:BX points to the allocation table for the current drive, DX has the number of allocation units, AL has the number of records per allocation unit, and CX has the size of the physical sector. At DS:[BX-1], the byte before the allocation table, is the dirty byte for the table. If set to 01, it means the table has been modified and must be written back to disk. If 00, the table is not modified. Any programs which get the address and directly modify the table must be sure to set this byte to 01 for the changes to be recorded. This byte should **never** be set to 00. Instead, perform a DISK RESET function (#0D hex) to write the table and reset the bit.

1C-20 Not used.

21 Random read. On entry, DS:DX points to an opened FCB. The current block and current record are set to agree with the random record field, then the record addressed by these fields is loaded at the current disk transfer address. If

end-of-file is encountered, AL returns either 01 or 03. If 01 is returned, no more data is available. If 03 is returned, a partial record is available, and it is filled out with zeros. A return of 02 means there is not enough room in the disk transfer segment to read one record, so the transfer is aborted. AL returns 00 if the transfer is completed successfully.

22 Random write. On entry, DS:DX points to an opened FCB. The current block and current record are set to agree with the random record field, then the record addressed by these fields is written (or, in the case of records not the same as sector sizes, it is buffered) from the disk transfer address. If the disk is full, AL returns 01. A return of 02 means there is not enough room in the disk transfer segment to write one record, so the transfer is aborted. AL returns 00 if the transfer is completed successfully.

23 File size. On entry, DS:DX points to an unopened FCB. The disk directory is searched for the first matching entry and if none is found, AL returns FF. Otherwise, the random record field is set with the size of the file (in terms of the record size field rounded up) and AL returns 00.

24 Set random record field. On entry, DS:DX points to an opened FCB. This function sets the random record field to the same file address as the current block and record fields.

25 Set vector. The interrupt type specified in AL is set to the 4-byte address DS:DX.

26 Create a new program segment. On entry, DX has a segment number at which to set up a new program segment. The entire 100 hex area at location zero in the current program segment is copied into location zero in the new program segment. The memory size information at location 6 is updated and the current termination and ^BREAK exit addresses are saved in the new program segment, starting at 0A hex.

27 Random block read. On entry, DS:DX points to an opened FCB, and CX contains a record count that must not be zero. The specified number of records (in terms of the record size field) are read from the file address specified by the random record field into the disk transfer address. If end-of-file is reached before all records have been read, AL returns either 01 or 03. A return of 01 indicates end-of-file and the last record is complete; a 03 indicates the last record is a partial record. If wrap-around above address FFFF hex in the disk transfer segment occurs, as many records as possible are read and AL returns 02. If all records are read successfully, AL returns 00. In any case, CX returns with the actual number of records read, and the random record field and the current block/record fields are set to address the next record.

28 Random block write. Essentially, the same as function 27 above, except for writing and a write-protect indication. If there is insufficient space on the disk, AL returns 01 and no records are written. If CX is zero upon entry, no records are written, but the file is set to the length specified by the random record field, whether longer or shorter than the current file size (allocation units are released or allocated as appropriate).

29 Parse file name. On entry, DS:SI points to a command line to parse, and ES:DI points to a portion of memory to be filled in with an unopened FCB. Leading tabs and spaces are ignored when scanning. If bit 0 of AL is equal to 1 on entry, then at most one leading file name separator will be ignored, along with any trailing tabs and spaces. The four filename separators are:

; , = +

If bit 0 of AL is equal to 1, then all parsing stops if a separator is encountered. The command line is parsed for a file name of the form *d:filename.ext*, and if found, a corresponding unopened FCB is created at ES:DI. The entry value of AL bits 1, 2, and 3 determine what to do if the drive, filename, or

extension, are missing. In each case, if the bit is a zero and the field is not present on the command line, then the FCB is filled with a fixed value (0, meaning the default drive for the drive field ; all blanks for the filename and extension fields). If the bit is a 1, and the field is not present on the command line, then that field in the destination FCB at ES:DI is left unchanged. If an asterisk (*) appears in the filename or extension, then all remaining characters in the name or extension are set to ?.

The following characters are illegal within DOS file specifications:

" / [ ] + = ; ,

Control characters and spaces also may not be given as elements of file specifications. If any of these characters are encountered while parsing, or the period (.) or colon (:) is found in an invalid position, then parsing stops at that point.

If either ? or * appears in the filename or extension, then AL returns 01; otherwise, it returns 00. DS:SI will return pointing to the first character after the filename.

2A Get date. Returns the date in CX:DX. CX has the year, DH has the month (1=Jan, 2=Feb, and so on), and DL has the day. If the time-of-day clock rolls over to the next day, the date will be adjusted accordingly, taking into account the number of days in each month and leap years.

2B Set date. On entry, CX:DX must have a valid date in the same format as returned by function 2A above. If the date is indeed valid and the set operation is successful, then AL returns 00. If the date is not valid, then AL returns FF.

2C Get time. Returns with time-of-day in CX:DX. Time is actually represented as four 8-bit binary quantities, as follows: CH has the hours (0-23), CL has minutes (0-59), DH has seconds (0-59), and DL has hundredths of seconds (0-99). This format is easily converted to a printable form yet can also be calculated upon (for example, subtracting two times).

2D Set time. On entry, CX:DX has time in the same format as returned by function 2C above. If any component of the time is not valid, the set operation is aborted and AL returns FF. If the time is valid, AL returns 00.

2E Set/reset verify flag. On entry, DL must be 0 and AL has the verify flag: 0 = no verify; 1 = verify after write. This flag is simply passed to the I/O system on each write, so its exact meaning is interpreted there.

DISK ERRORS

If a disk error occurs at any time during any command or program, DOS retries the operation three times. If the operation cannot be completed successfully, DOS returns an error message in the following format:

type ERROR WHILE *I/O action* ON DRIVE *d:*
Abort, Ignore, Retry: _

In this message, *type* may be one of the following:

Write Protect
Not Ready
Seek
Data
Sector Not Found
Write Fault
Disk

The *I/O action* may be either of the following:

Reading
Writing

The drive *d:* indicates the drive in which the error has occurred.

DOS waits entry of one of the following responses:

A Abort. Terminates the program requesting the disk read or write.

I Ignore. Ignores the bad sector and pretends the error did not occur.

R Retry. Repeats the operation. This response is particularly useful if the operator has corrected the error (such as with Not Ready or Write Protect).

Usually, you will want to attempt recovery by entering responses in the order:

R To try again.
A To terminate the program and try a new diskette.

One other error message might be related to faulty disk read or write:

FILE ALLOCATION TABLE BAD FOR DRIVE *d:*

This message means that the copy in memory of one of the allocation tables has pointers to nonexistent blocks. Possibly, the diskette was not formatted before use.

INDEX

A

B

C

D

E

F

I

L

N

P

R

S

W

Notes:

DISK OPERATING SYSTEM REFERENCE GUIDE COMMENTS FORM

Please assist us by answering the questions listed below. Then detach this prepaid comments form and drop it into a mailbox. This will help us to attain our goal, which is to provide you with documentation of the highest quality.

CHECK ☑ THE APPROPRIATE ANSWER:	YES	NO
• Is this document easy to read?	☐	☐
• Does the text flow in a logical manner?	☐	☐
• Are the directions easy to follow?	☐	☐
• Are the examples and syntax statements clear and usable?	☐	☐
• Are the technical terms clearly defined?	☐	☐
• Did you find any inconsistencies or errors in the manual?	☐	☐
• Do you feel that this document provides all the information that you need to understand and to use the Disk Operating System?	☐	☐

COMMENTS: _____

_____

_____

_____

_____

Thank you for taking the time to assist us.

(Optional)

NAME: _____

ADDRESS: _____

CITY/STATE: _____

PHONE NO: _____

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 400 HOUSTON, TEXAS USA

POSTAGE WILL BE PAID BY ADDRESSEE

COMPAQ Computer Corporation
12330 Perry Road
Houston, Texas 77070

Fold here

Tape Please do not staple Tape